รหัสโครงการ SUT7-711-45-12-35

รายงานการวิจัย

การบีบอัดสัญญาณภาพถ่ายจากดาวเทียมโดยการแปลงเวฟเล็ต และการศึกษาเทคนิคการบีบอัดสัญญาณภาพของ JPEG2000

Satellite image compression using the discrete wavelet transform and a study of JPEG2000

คณะผู้วิจัย

หัวหน้าโครงการ

ผู้ช่วยศาสตราจารย์ ดร.กิตติ อัตถกิจมงคล

สาขาวิชาวิศวกรรมไฟฟ้า

สำนักวิชาวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ผู้ร่วมวิจัย

นายณัฐนันท์ ทัดพิทักษ์กุล

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2545

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

ธันวาคม 2546

กิตติกรรมประกาศ

การวิจัยครั้งนี้ได้รับทุนอุดหนุนการวิจัยจากมหาวิยาลัยเทคโนโลยีสุรนารีปีงบประมาณ 2545 คณะผู้วิจัยขอขอบคุณ อ. ศิริพงษ์ วงษ์คาร จาก มหาวิทยาลัยเทคโนโลยีมหานคร ที่ได้เอื้อเฟื้อภาพถ่ายจากดาวเทียมเพื่อใช้ในงานวิจัย ขอขอบคุณผู้จัดทำเวปไซท์ http://visibleearth.nasa.gov/ ขององค์การ NASA ที่เผยแพร่ภาพถ่ายจากดาวเทียมและยินดีให้นักวิจัยนำภาพเหล่านั้นมาศึกษาและวิจัยต่อไป

บทคัดย่อ

ในช่วงเวลาไม่กี่ปีที่ผ่านมา มีการพัฒนาไปอย่างรวดเร็วของเครื่องคอมพิวเตอร์และวงจรรวม ทำให้การใช้สัญญาณภาพในรูปแบบดิจิตอลได้เป็นที่นิยมแพร่หลาย เหตุผลสำคัญเนื่องจากสัญญาณภาพในรูปแบบดิจิตอลสามารถดัดแปลง แก้ไขได้ง่าย ในปัจจุบันการจัดเก็บหรือการส่งสัญญาณภาพนิยมใช้ภาพในรูปแบบดิจิตอล แต่ปัญหาสำคัญในการใช้สัญญาณภาพในรูปแบบดิจิตอลคือ ภาพแต่ละภาพต้องใช้เนื้อที่จำนวนมากในการจัดเก็บข้อมูลลงในอุปกรณ์เก็บข้อมูล หรือต้องใช้เวลานานในการส่งข้อมูลภาพจากที่หนึ่งไปยังอีกที่หนึ่งผ่านสายส่ง อย่างไรก็ตามจากการศึกษาวิจัยพบว่า ในภาพทั่วๆไป จะมีข้อมูลบางส่วนของภาพที่ไม่จำเป็นหรือซ้ำซ้อนกันซึ่งเราสามารถใช้ประโยชน์จากข้อมูลส่วนนี้ในการลดขนาดข้อมูลของภาพให้มีขนาดเล็กลงแต่ยังคงคุณภาพของภาพไว้ เราเรียกกระบวนการนี้ว่า การบีบอัดสัญญาณภาพ ซึ่งทำให้การใช้สัญญาณภาพในรูปแบบดิจิตอลยิ่งเป็นที่แพร่หลาย โดยเฉพาะ การใช้ฐานข้อมูลของสัญญาณภาพ หรือระบบการส่งสัญญาณภาพ ในงานวิจัยนี้เราได้ศึกษาและเสนอขั้นตอนของการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมโดยใช้การแปลงเวฟเล็ต เนื่องจากในปัจจุบันเราใช้ภาพถ่ายจากดาวเทียมในงานหลายๆด้าน เช่น การตรวจดูแหล่งทรัพยากรธรรมชาติ การพยากรณ์อากาศ ทำให้ฐานข้อมูลของภาพถ่ายดาวเทียมและการส่งสัญญาณภาพถ่ายจากดาวเทียมต้องการขบวนการบีบอัดสัญญาณภาพที่มีประสิทธิภาพ นอกจากนี้เรายังได้ศึกษาเทคนิคพื้นฐานของมาตราฐานการเข้ารหัสสัญญาณภาพใหม่ที่ชื่อว่า JPEG2000 ซึ่งใช้การแปลงเวฟเล็ตเช่นกัน โดยมาตราฐานใหม่นี้ได้แสดงให้เห็นว่ามีข้อดีมากกว่ามาตราฐานเดิมของ JPEG

ABSTRACT

Due to the advent of the digital computer and subsequence development of the advanced integrated circuit, the demand for handling images in digital form has increased dramatically in the past decade. The reason for this interest in digital image is that representing image in digital form allows visual information to be easily manipulated in useful ways. However, in spite of this advantage, there is one potential problem with digital images: the large number of bits required to represent them. Fortunately, most visual images contains a large amount of statistical redundancy and visual irrelevancy that one can take advantage of to reduce the number of bits and make widespread use of digital imagery practical. For this reason, image compression becomes essential for many applications such as image databases and image transmission. In this research, we study the compression algorithm for satellite images based on the discrete wavelet transform. Since the satellite remote sensing data is extensively used in many applications such as natural resource monitoring and weather forecast system, there is a growing need for satellite image compression. In addition, we also study the fundamental of the latest international image compression standard: the JPEG2000. This new standard is also based on the wavelet transform and has been shown to outperform the previous standard from the JPEG committee.

สารบัญ

หน้า

สารบัญ(ต่อ)

สารบัญตาราง

ตาราง หน้า

สารบัญภาพ

ภาพ หน้า

สารบัญภาพ (ต่อ)

ภาพ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันมีการใช้ดาวเทียมในการถ่ายภาพ เพื่อใช้ในงานต่างๆ เพิ่มมากขึ้นเช่น การตรวจสอบและสำรวจการเปลี่ยนแปลงของทรัพยากรธรรมชาติ ตลอดจนพยากรณ์อากาศ ในการใช้งานสัญญาณภาพถ่ายดาวเทียมในงานบางประเภท จำเป็นที่จะต้องแปลความหมายของสัญญาณภาพที่ได้ ดังนั้นจึงจำเป็นที่จะต้องการใช้ภาพถ่ายหลายๆ ภาพในการแปลความหมาย ดังนั้นในการส่งสัญญาณภาพมายังสถานีภาคพื้นดิน จึงต้องใช้เวลาเพิ่มมากขึ้น แต่ในขณะที่เวลาในการส่งสัญญาณมายังสถานีภาคพื้นดินยังคงเท่าเดิม

เพื่อเป็นการแก้ปัญหานี้ งานวิจัยฉบับนี้จึงได้เสนอวิธีการลดการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมโดยการแปลงเวฟเล็ต และทำการศึกษาเทคนิคการบีบอัดสัญญาณภาพของ JPEG2000

1.2 วัตถุประสงค์ของการวิจัย

1. เพื่อศึกษาและออกแบบอัลกอริทึมการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม โดยใช้วิธีการแปลงเวฟเล็ต (wavelet transform) และการพัฒนาอัลกอริทึม Set Partitioning in Hierarchical Trees (SPIHT) ในการเข้ารหัส และการถอดรหัส
2. เพื่อหาข้อสรุปในการใช้งานที่เหมาะสมของการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม โดยใช้วิธีการแปลงเวฟเล็ต และใช้อัลกอริทึมที่ได้พัฒนา ในการเข้ารหัส และการถอดรหัส
3. เพื่อเป็นแนวทางในการพัฒนาขบวนการบีบอัดสัญญาณภาพ ให้กับผู้ที่สนใจได้ทำการศึกษาและพัฒนาให้ดียิ่งขึ้น

1.3 ข้อตกลงเบื้องต้น

1. อัลกอริทึมการบีบอัดสัญญาณภาพ ที่นำมาพัฒนาคือ SPIHT (Amir and William, 1996)
2. โปรแกรมการบีบอัดสัญญาณภาพ เขียนด้วยโปรแกรม Visual C++

1.4 ขอบเขตของการวิจัย

1. พัฒนาอัลกอริทึม SPIHT ให้มีประสิทธิภาพการบีบอัดสัญญาณภาพเพิ่มขึ้น
2. พัฒนาโปรแกรมการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม โดยใช้วิธีการแปลงเวฟเล็ต และใช้อัลกอริทึมที่ได้พัฒนาในการเข้ารหัส และการถอดรหัส

1.5 ขั้นตอนการดำเนินงาน

1. ดำเนินการศึกษาการแปลงสัญญาณภาพ ด้วยวิธีการแปลงเวฟเล็ตแบบดีสครีท (discrete wavelets transform: DWT) แบบ 2 มิติ ใช้หลักการแยกองค์ประกอบเป็นแบนด์ย่อย (subband decomposition)
2. ดำเนินการศึกษาการลดสัญญาณภาพด้วยวิธี SPIHT
3. ดำเนินการพัฒนาอัลกอริทึมการบีบอัดสัญญาณภาพด้วยวิธี SPIHT

1.6 ประโยชน์ที่ได้รับจากการวิจัย

1. ได้เรียนรู้ทฤษฎีและการประยุกต์ใช้การแปลงเวฟเล็ต ในการบีบอัดสัญญาณภาพ
2. ได้อัลกอริทึมการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม โดยใช้วิธีการแปลงเวฟเล็ต
3. ได้ระดับการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมที่เหมาะสม โดยใช้วิธีการที่ได้นำเสนอ
4. ได้เป็นแนวทางในการพัฒนาขบวนการบีบอัดสัญญาณภาพ ให้กับผู้สนใจได้ทำการศึกษาและพัฒนาให้ดียิ่งขึ้นต่อไปในอนาคต

บทที่ 2
การแปลงเวฟเล็ต

2.1 บทนำ

บทนี้จะกล่าวถึงความเป็นมาของการแปลงเวฟเล็ต (wavelet transform: WT) ทฤษฎีพื้นฐานของการแปลงเวฟเล็ต การแปลงเวฟเล็ตแบบดีสครีท (discrete wavelet transform: DWT) แบบ 1 มิติ ด้วยหลักการฟิลเตอร์แบงค์ (filter bank) และการแปลง DWT แบบ 2 มิติ ด้วยหลักการแยกองค์ประกอบเป็นแบนด์ย่อย (subband decomposition)

2.2 การแปลงเวฟเล็ตแบบดีสครีทใน 1 มิติ

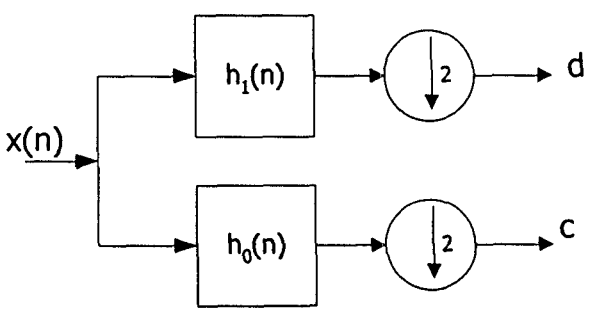
การแปลงเวฟเล็ตแบบดีสครีทแบบ 1 มิติ โดยใช้หลักการของฟิลเตอร์แบงค์ (filter bank) (Olivier, and Martin, 1991) มีหลักการวิเคราะห์ (analysis) สัญญาณด้วยฟิลเตอร์แบงค์เป็นดังภาพที่ 2.1 กำหนดให้ x(n) คือข้อมูลอินพุต $h_0(n)$ คือตัวฟิลเตอร์ที่กรองความถี่ต่ำ และ $h_1(n)$ คือตัวฟิลเตอร์ที่กรองความถี่สูง และมีแผนภาพการลดข้อมูลลงครึ่งหนึ่ง (downsampling) เป็นดังภาพที่ 2.2 สามารถเขียนเป็น สมการการวิเคราะห์ได้ดังสมการที่ 2.1 และ 2.2

$$C(k) = \sum_m h_0(2k-m)x(m) \tag{2.1}$$

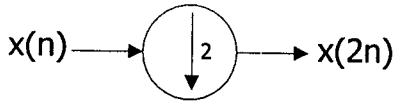
$$d(k) = \sum_m h_1(2k-m)x(m) \tag{2.2}$$

เมื่อ c คือ สัมประสิทธิ์เวฟเล็ตขององค์ประกอบความถี่ต่ำ

d คือ สัมประสิทธิ์เวฟเล็ตขององค์ประกอบความถี่สูง

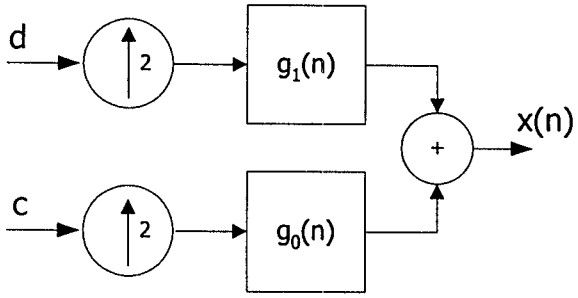


ภาพที่ 2.1 แผนภาพการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์ แบบ 1 มิติ

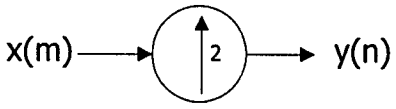


ภาพที่ 2.2 แผนภาพการลดข้อมูลลงครึ่งหนึ่ง

และมีหลักการสังเคราะห์ (synthesis) สัญญาณด้วยฟิลเตอร์แบงค์เป็นดังภาพที่ 2.3 กำหนดให้ $g_0(n)$ คือ ตัวฟิลเตอร์ที่กรองความถี่ต่ำ และ $g_1(n)$ คือ ตัวฟิลเตอร์ที่กรองความถี่สูง และมีแผนภาพการเพิ่มข้อมูลขึ้นเป็นสองเท่า (upsampling) เป็นดังภาพที่ 2.4 สามารถเขียนเป็นสมการการสังเคราะห์ได้ดังสมการที่ 2.10



ภาพที่ 2.3 แผนภาพการสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์ แบบ 1 มิติ



$$y(n) = x(l), \quad n = 2l$$
$$= 0 \quad , \text{อื่นๆ}$$

ภาพที่ 2.4 แผนภาพการเพิ่มข้อมูลขึ้นเป็นสองเท่า

การวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบหลายแบนด์ย่อย สามารถทำได้โดยการนำเอาการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 1 มิติแบบ 1 แบนด์ย่อยมาประยุกต์ใช้ได้ดังสมการที่ 2.3 และ 2.4 และมีแผนภาพการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบหลายแบนด์ย่อยดังภาพที่ 2.5 ซึ่งภาพที่ 2.5 เป็นการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 3 แบนด์ย่อย

$$C_j(k) = \sum_m h_0(2k-m)c_{j+1}(m) \tag{2.3}$$

$$d_j(k) = \sum_m h_1(2k-m)c_{j+1}(m) \tag{2.4}$$

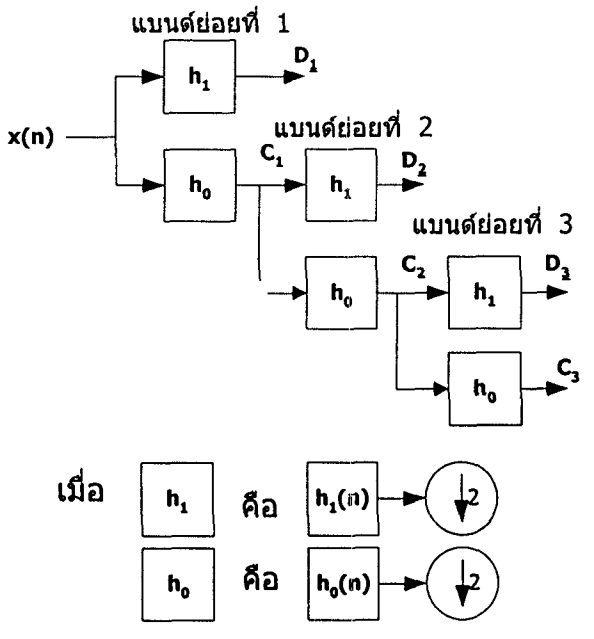
เมื่อ c_j คือ สัมประสิทธิ์เวฟเล็ตขององค์ประกอบความถี่ต่ำ ที่แบนด์ย่อยที่ j

d_j คือ สัมประสิทธิ์เวฟเล็ตขององค์ประกอบความถี่สูง ที่แบนด์ย่อยที่ j

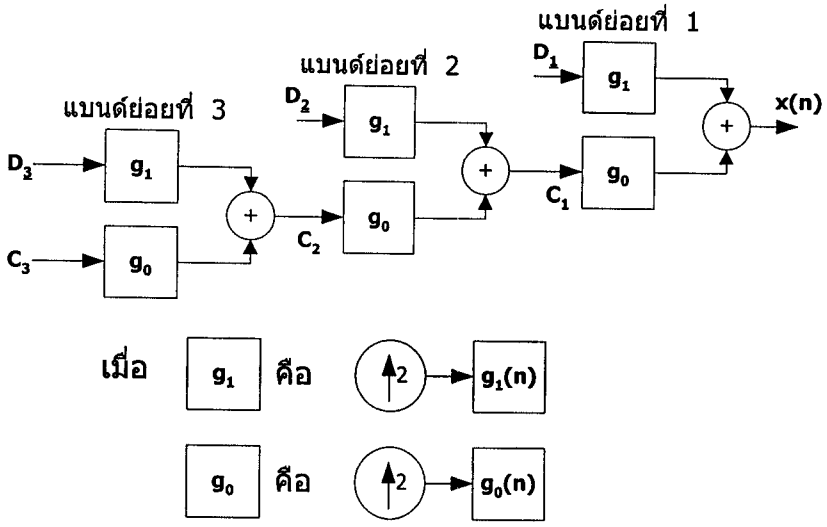
การสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบหลายแบนด์ย่อย สามารถทำได้โดยการนำเอาการสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 1 มิติ แบบ 1 แบนด์ย่อย มาประยุกต์ใช้ได้ดังสมการที่ 2.5 และ 2.6 และมีแผนภาพสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบหลายแบนด์ย่อยดังภาพที่ 2.6 ซึ่งภาพที่ 2.6 เป็นการสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 3 แบนด์ย่อย

$$x(n) = \sum_m c_1(m)g_0(n-2m) + \sum_m d_1(m)g_1(n-2m) \tag{2.5}$$

$$c_{j+1}(k) = \sum_p c_j(p)g_0(k-2p) + \sum_p d_j(p)g_1(k-2p) \tag{2.6}$$

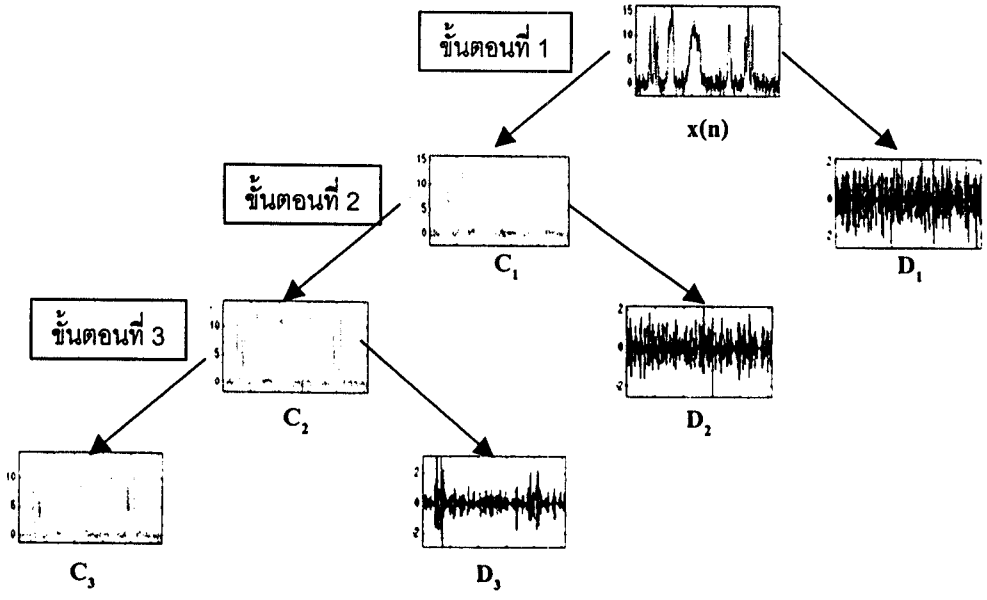


ภาพที่ 2.5 แผนภาพการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์ แบบ 3 แบนด์ย่อย



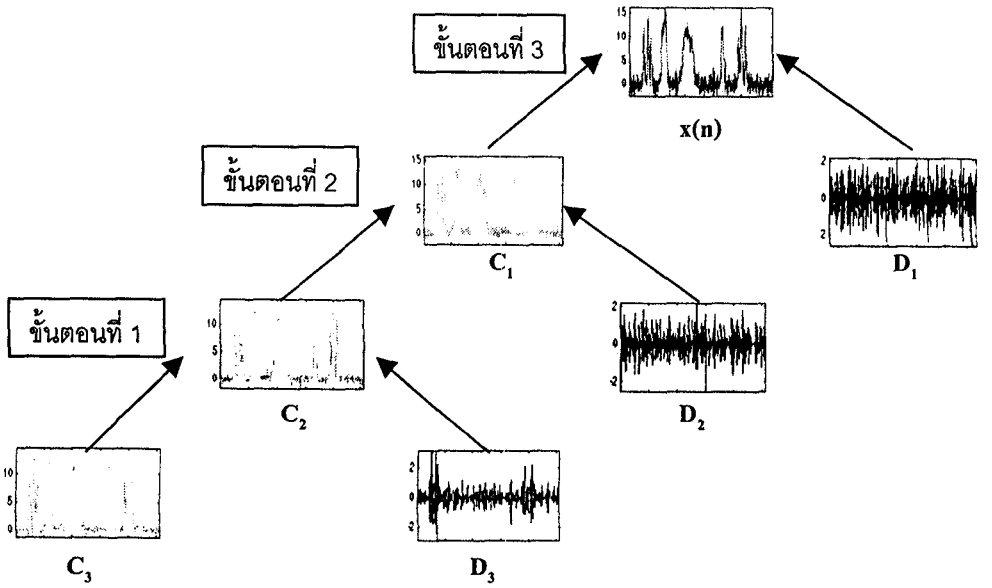
ภาพที่ 2.6 แผนภาพการสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์ แบบ 3 แบนด์ย่อย

มีตัวอย่างการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 3 แบนด์ย่อยเป็นดังภาพที่ 2.7 และมีตัวอย่างการสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 3 แบนด์ย่อยเป็นดังภาพที่ 2.8



ภาพที่ 2.7 ตัวอย่างการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 3 แบนด์ย่อย

(Michel et al., 1996)

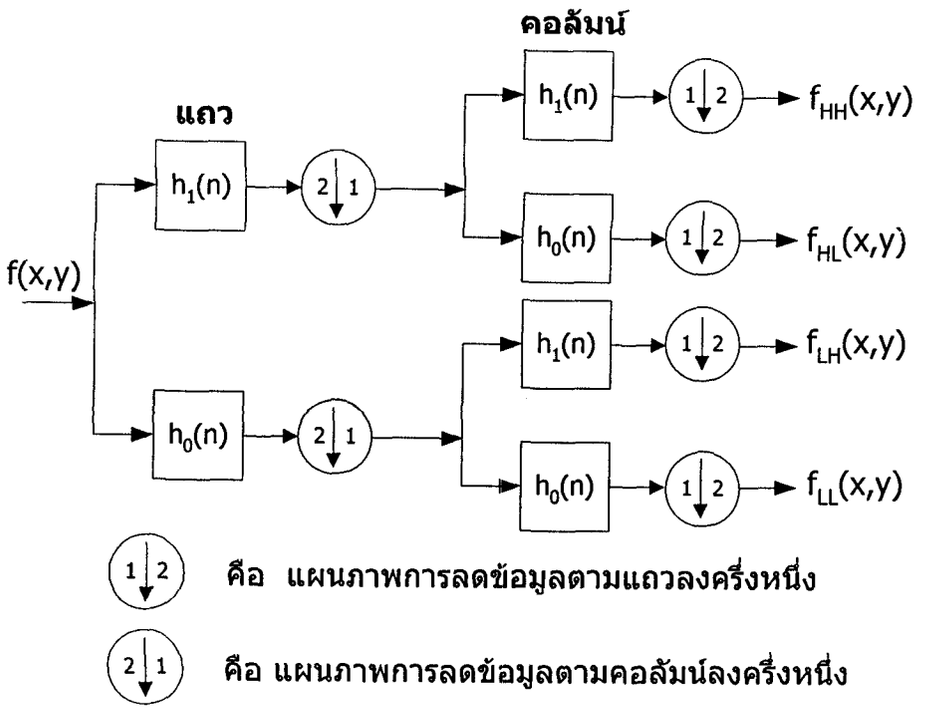


ภาพที่ 2.8 ตัวอย่างการสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์แบบ 3 แบนด์ย่อย (Michel et al., 1996)

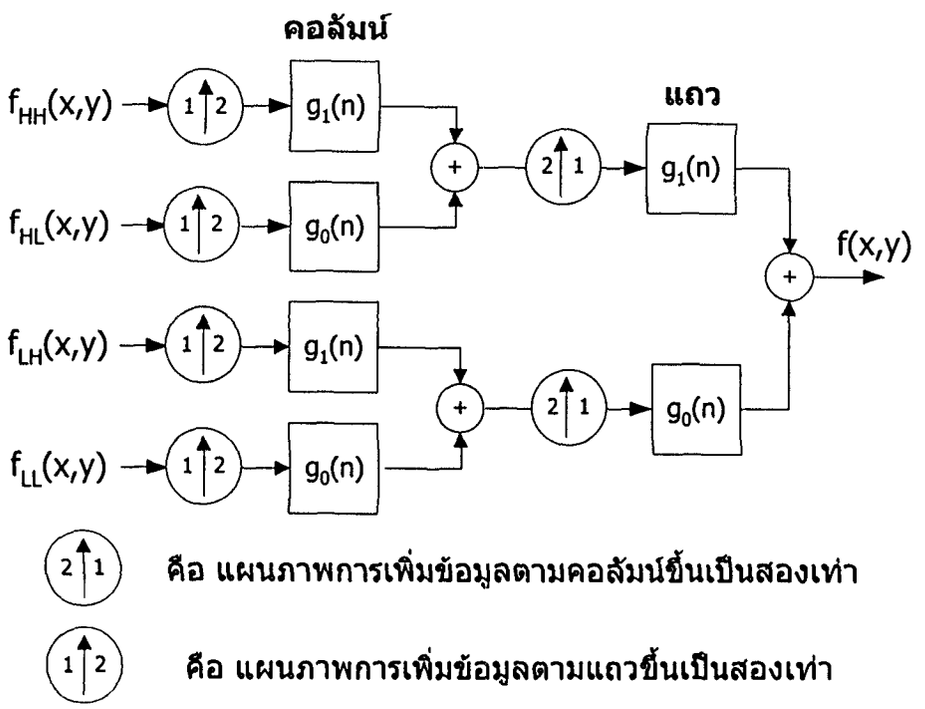
2.3 การแปลงเวฟเล็ตแบบดีสครีทใน 2 มิติ

การแปลงเวฟเล็ตแบบดีสครีทแบบ 2 มิติ ใช้หลักการแยกองค์ประกอบเป็นแบนด์ย่อย (subband decomposition) (Stephane, 1989; Marc, Michel, Pierre, and Ingrid, 1992) มีหลักการประมวลผลภาพ โดยใช้การแปลงเวฟเล็ตออกเป็นสองขั้นตอนคือ การแยกองค์ประกอบด้วยเวฟเล็ต มีวิธีการแปลงเป็นดังภาพที่ 2.9 กำหนดให้ f(x,y) คือภาพต้นแบบ $f_{LL}(x,y)$, $f_{LH}(x,y)$, $f_{HL}(x,y)$ และ $f_{HH}(x,y)$ คือสัมประสิทธิ์เวฟเล็ต และการสร้างกลับจากองค์ประกอบด้วยเวฟเล็ต เป็นดังภาพที่ 2.10

จากภาพที่ 2.9 ภาพต้นแบบจะถูกกรองไปตามแนวแถวด้วยฟิลเตอร์ $h_0(n)$ และ $h_1(n)$ จากนั้นนำผลที่ได้มาลดจำนวนข้อมูลลงครึ่งหนึ่ง โดยการเก็บข้อมูล 1 คอลัมน์จากข้อมูล 2 คอลัมน์ จะได้เอาต์พุตความถี่สูงและความถี่ต่ำ จากนั้นเอาต์พุตทั้งสองจะถูกกรองตามแนวคอลัมน์ด้วยฟิลเตอร์ $h_0(n)$ และ $h_1(n)$ และลดจำนวนข้อมูลลงครึ่งหนึ่ง โดยการเก็บข้อมูลมา 1 แถวจากข้อมูล 2 แถว จะได้ภาพของสัมประสิทธิ์แบนด์ย่อยความถี่ต่ำคือ $f_{LL}(x,y)$ จำนวนหนึ่งภาพ และภาพของสัมประสิทธิ์แบนด์ย่อยความถี่สูงจำนวน 3 ภาพคือ $f_{HL}(x,y)$, $f_{LH}(x,y)$ และ $f_{HH}(x,y)$ โดยแต่ละภาพจะมีขนาดลดลงเป็นหนึ่งในสี่ของภาพต้นแบบ



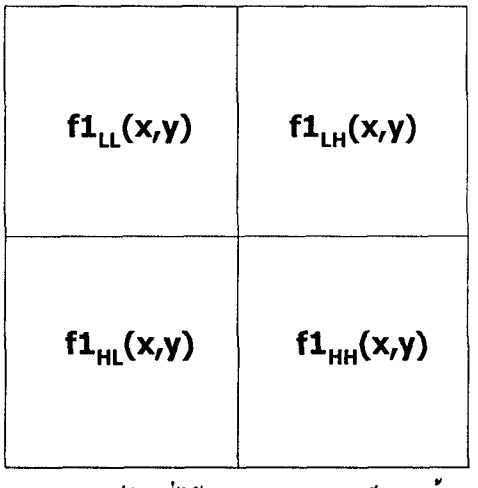
ภาพที่ 2.9 แผนภาพการแยกองค์ประกอบด้วยเวฟเล็ต



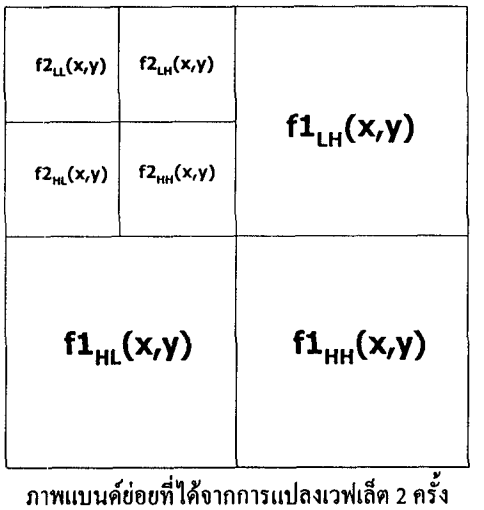
ภาพที่ 2.10 แผนภาพการสร้างกลับจากองค์ประกอบด้วยเวฟเล็ต

จากภาพที่ 2.10 การสร้างภาพขึ้นมาใหม่จากองค์ประกอบแบนด์ย่อย เริ่มจากภาพของสัมประสิทธิ์แบนด์ย่อยทั้ง 4 ภาพจะถูกเพิ่มจำนวนข้อมูลเป็น 2 เท่า โดยการเติมศูนย์เข้าไปในระหว่างแต่ละคู่ของแถว หลังจากเพิ่มจำนวนข้อมูลแล้วผลลัพธ์ที่ได้จะถูกกรองไปตามคอลัมน์ด้วยฟิลเตอร์ $g_0(n)$ และ $g_1(n)$ และบวกเข้าด้วยกันตามภาพที่ 2.10 จากนั้นนำผลลัพธ์ที่ได้มาเติมศูนย์เข้าไปในระหว่างแต่ละคู่ของคอลัมน์ หลังจากเพิ่มจำนวนข้อมูลแล้วผลลัพธ์ที่ได้จะถูกกรองไปตามแถวด้วยฟิลเตอร์ $g_0(n)$ และ $g_1(n)$ และบวกเข้าด้วยกันตามภาพที่ 2.10 ผลลัพธ์ที่ได้คือภาพต้นแบบที่สร้างกลับขึ้นมาใหม่

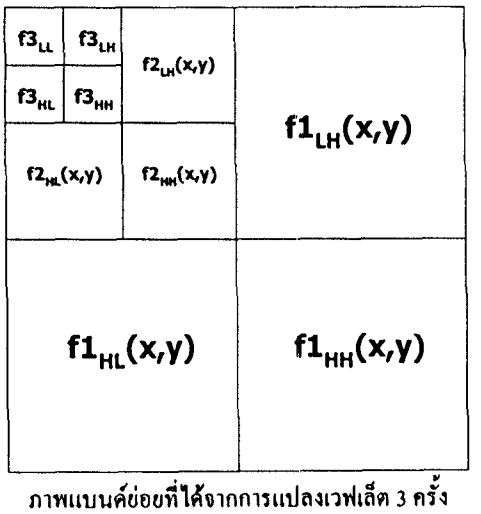
การแปลงเวฟเล็ตของข้อมูลภาพแบบหลายครั้ง เป็นดังภาพที่ 2.11 ซึ่งภาพที่ 2.11 จะแสดงภาพแบนด์ย่อยที่ผ่านการแปลงเวฟเล็ตหลายครั้ง และมีตัวอย่างภาพที่ผ่านการแปลงเวฟเล็ตด้วยวิธีการแยกองค์ประกอบด้วยเวฟเล็ต 3 ครั้ง เป็นดังภาพที่ 2.12



ภาพแบนด์ย่อยที่ได้จากการแปลงเวฟเล็ต 1 ครั้ง

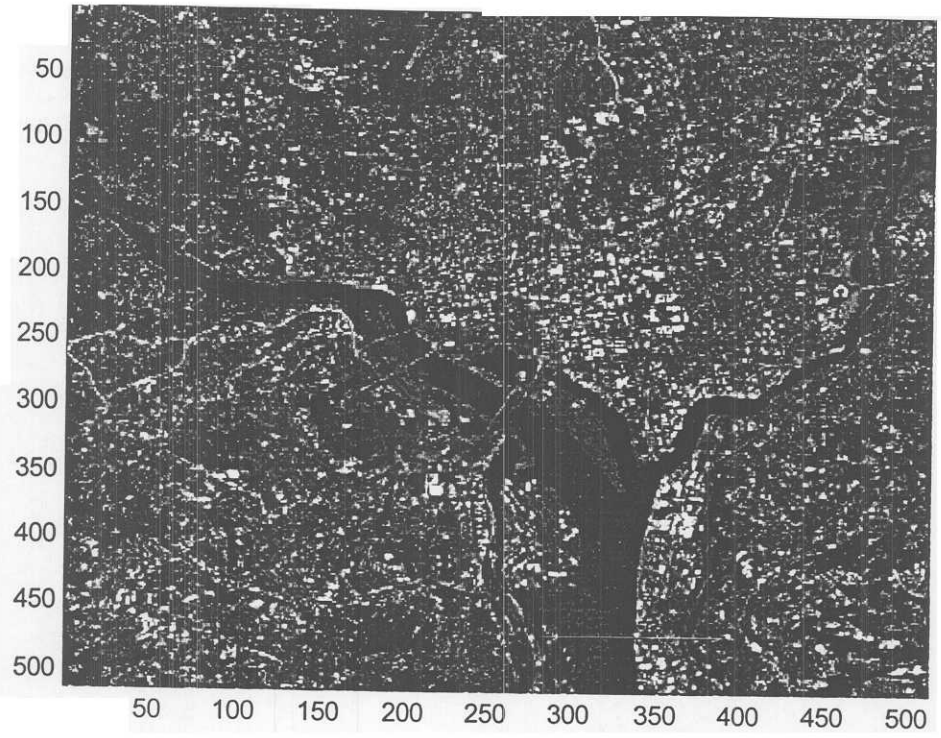


ภาพแบนด์ย่อยที่ได้จากการแปลงเวฟเล็ต 2 ครั้ง

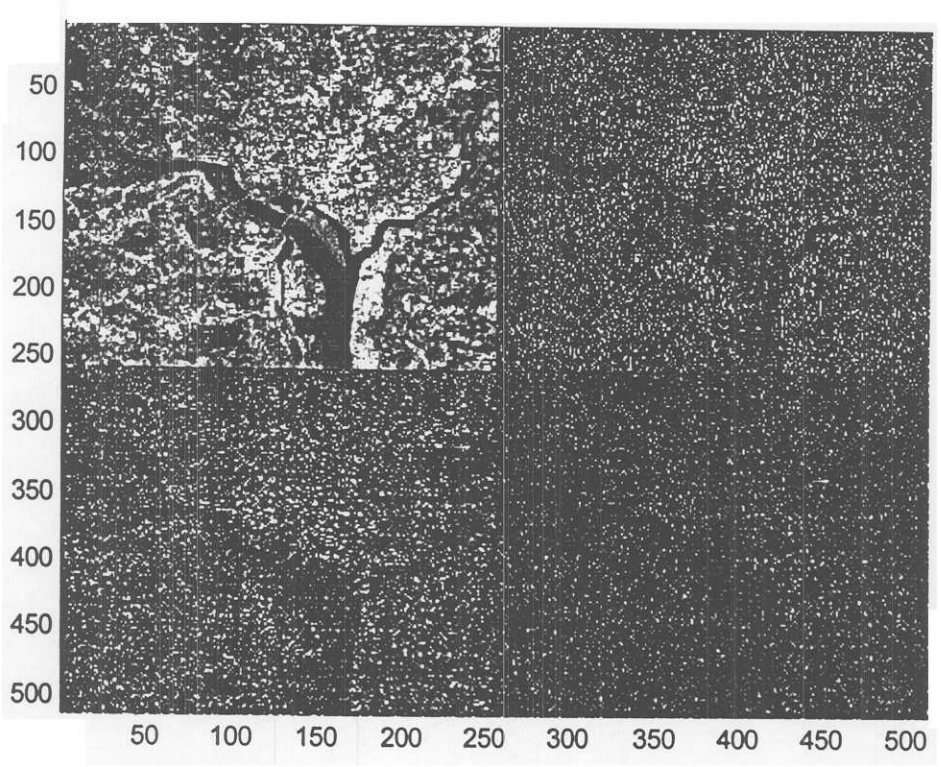


ภาพแบนด์ย่อยที่ได้จากการแปลงเวฟเล็ต 3 ครั้ง

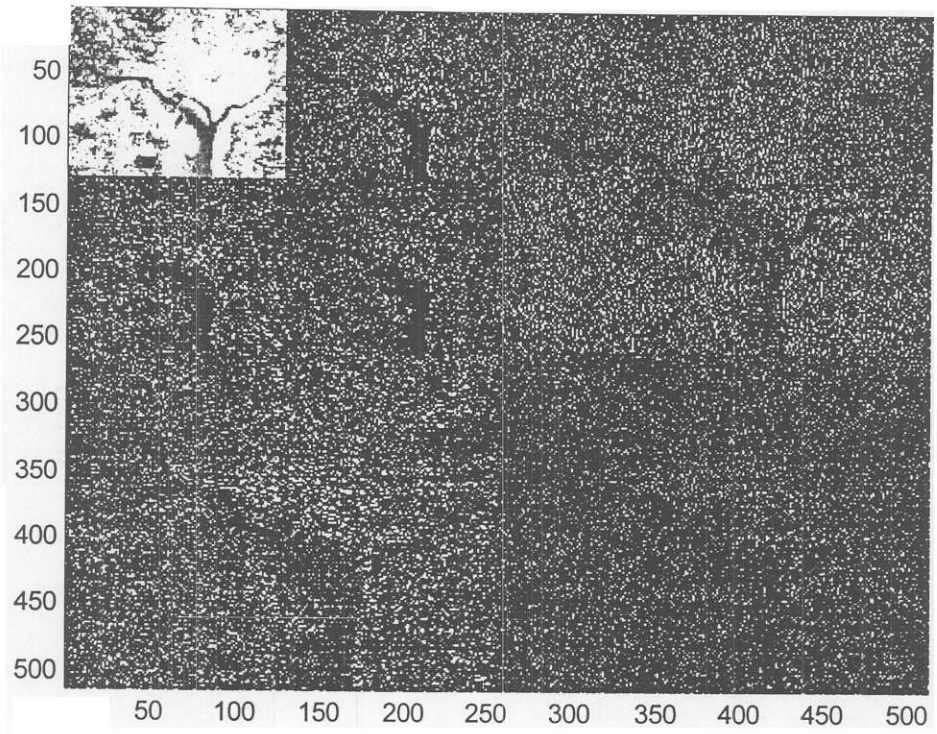
ภาพที่ 2.11 แสดงภาพแบนด์ย่อยที่ผ่านการแปลงเวฟเล็ตหลายครั้ง



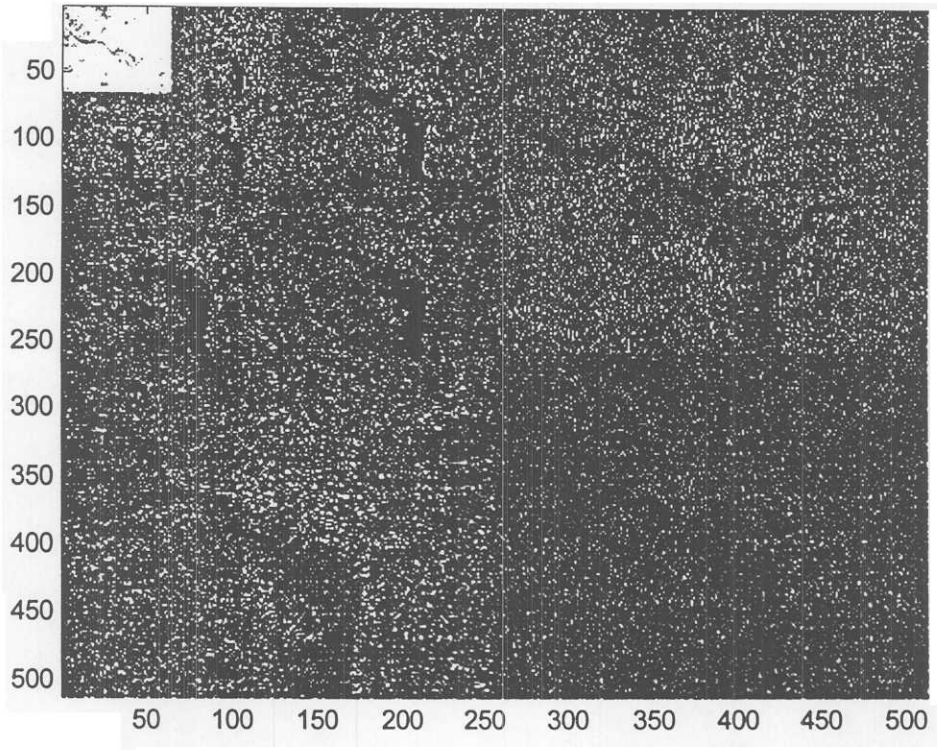
ภาพที่ 2.12.a ภาพต้นแบบ (ขนาด 512x512x8 บิต)



ภาพที่ 2.12.b ภาพที่ผ่านการแปลงเวฟเล็ตด้วยวิธีการแยกองค์ประกอบด้วยเวฟเล็ต 1 ครั้ง



ภาพที่ 2.12.c ภาพที่ผ่านการแปลงเวฟเล็ตด้วยวิธีการแยกองค์ประกอบด้วยเวฟเล็ต 2 ครั้ง



ภาพที่ 2.12.d ภาพที่ผ่านการแปลงเวฟเล็ตด้วยวิธีการแยกองค์ประกอบด้วยเวฟเล็ต 3 ครั้ง

ภาพที่ 2.12 ตัวอย่างภาพที่ผ่านการแปลงเวฟเล็ตด้วยวิธีการแยกองค์ประกอบด้วยเวฟเล็ต

2.4 สรุป

ในบทนี้ได้กล่าวถึงการแปลงเวฟเล็ตดีสครีทใน 1 มิติ ด้วยวิธีการฟิลเตอร์แบงค์ โดยการแปลงเวฟเล็ตใช้วิธีการวิเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์และการแปลงกลับเวฟเล็ตใช้วิธีสังเคราะห์สัญญาณด้วยฟิลเตอร์แบงค์ และการแปลงเวฟเล็ตแบบดีสครีทใน 2 มิติ โดยการแปลงเวฟเล็ตใช้วิธีการแยกองค์ประกอบด้วยเวฟเล็ต และการแปลงกลับเวฟเล็ตใช้วิธีการสร้างกลับจากองค์ประกอบด้วยเวฟเล็ต

บทที่ 3

การบีบอัดสัญญาณภาพถ่ายจากดาวเทียมด้วยอัลกอริทึม

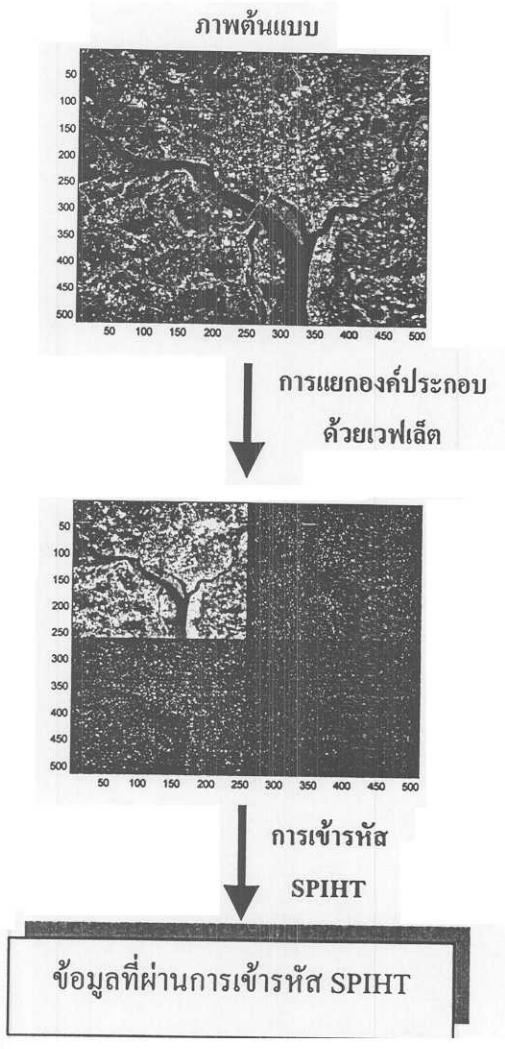
Set Partitioning in Hierarchical Tree (SPIHT)

3.1 บทนำ

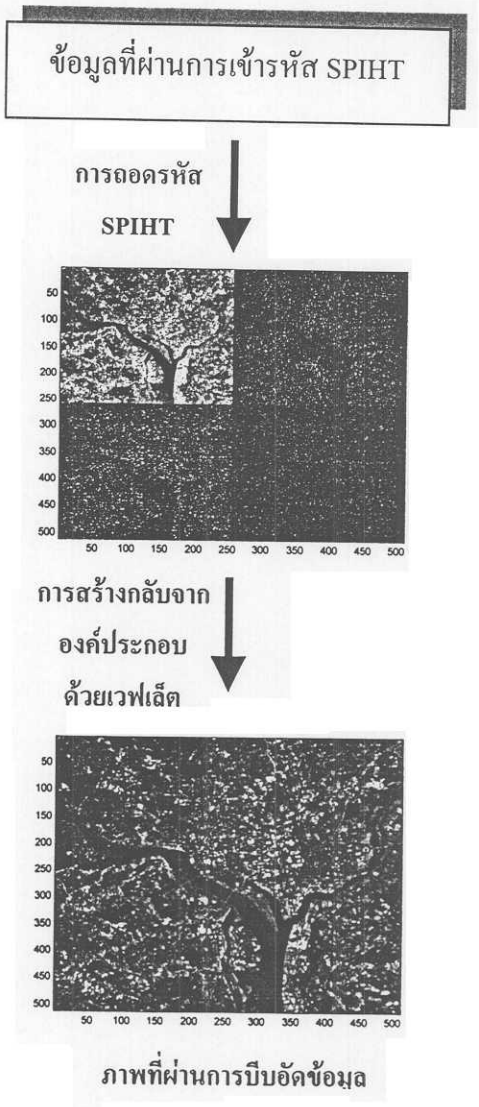
ภาพถ่ายจากดาวเทียมเมื่อนำมาแปลงด้วยวิธีเวฟเล็ตแบบดีสครีท สัญญาณที่เป็นพื้นผิววัตถุจะอยู่ที่สัมประสิทธิ์ที่แบนด์ย่อยความถี่ต่ำ และสัญญาณที่เป็นขอบหรือลายเส้นของวัตถุจะอยู่ที่สัมประสิทธิ์ที่แบนด์ย่อยความถี่สูง ดังนั้นการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมด้วยการแปลงเวฟเล็ตจะต้องให้ความสำคัญกับสัมประสิทธิ์ทุกแบนด์ย่อย เพื่อทำให้ภาพที่ผ่านการบีบอัดสัญญาณมีรายละเอียดของภาพเหมือนภาพต้นแบบมากที่สุด อัลกอริทึม SPIHT เป็นอัลกอริทึมหนึ่งที่เหมาะสม เนื่องจากอัลกอริทึมนี้เข้ารหัสโดยให้ความสำคัญกับขนาดของสัมประสิทธิ์ และไม่สนใจว่าสัมประสิทธิ์ตัวนั้นจะอยู่ในระดับแบนด์ย่อยใด

3.2 อัลกอริทึม SPIHT

Embedded Zerotrees Wavelet (EZW) วิธีนี้นำเสนอครั้งแรกโดย Jerome M. Shapiro (1993) ให้อัตราการลดสัญญาณที่ดีกว่าการบีบอัดสัญญาณภาพด้วยวิธี JPEG ซึ่งเป็นวิธีที่ได้รับการพัฒนาขึ้นมาจากสองหน่วยงานคือ International Organization for Standardization (ISO) และ International Telecommunication Union (ITU) และสามารถนำไปใช้ได้กับรูปภาพทุกประเภท โดยไม่ต้องปรับแก้อัลกอริทึม เพราะอัลกอริทึมนี้จะเข้ารหัสข้อมูลแบบปรับเปลี่ยนได้เอง (adaptive coding) (Jerome, 1993) นอกจากนั้นการลดสัญญาณภาพแบบ EZW ยังไม่เกิดปัญหาเรื่องการเกิดบล็อคอาติแฟคต์ (block artifacts) คือการเกิดบล็อคที่บริเวณขอบของวัตถุและลายเส้นอย่างชัดเจน (ศิริพร เดชะศิลารักษ์, 2543; สุขสันต์ จิรเชวง และวุฒิพงศ์ อารกุล 2544) เพราะ EZW จะเข้ารหัสโดยให้ความสำคัญกับขนาดของสัมประสิทธิ์ และไม่สนใจว่าสัมประสิทธิ์ตัวนั้นจะอยู่ในระดับความถี่ใด (ขวัญฤทัย ไพรีพ่ายฤทธิ์, 2542) ได้มีการพัฒนาอัลกอริทึมการเข้ารหัสด้วย EZW ให้มีประสิทธิภาพเพิ่มขึ้นมากมายหลายวิธี แต่ในงานวิจัยนี้เลือกใช้อัลกอริทึมการเข้ารหัสด้วยวิธี SPIHT ซึ่งให้อัตราการบีบอัดสัญญาณที่ดีกว่าการบีบอัดสัญญาณภาพด้วยวิธี EZW (Amir and William, 1996; Al, 2000) นอกจากนั้นการบีบอัดสัญญาณภาพด้วยวิธี SPIHT เป็นอัลกอริทึมที่เร็วและมีประสิทธิภาพ (Brian, and Thomas, 2001) มีขั้นตอนการเข้ารหัส (encode) เป็นดังภาพที่ 3.1.a และมีขั้นตอนการถอดรหัส (decode) เป็นดังภาพที่ 3.1.b



ภาพที่ 3.1.a ขั้นตอนการเข้ารหัส



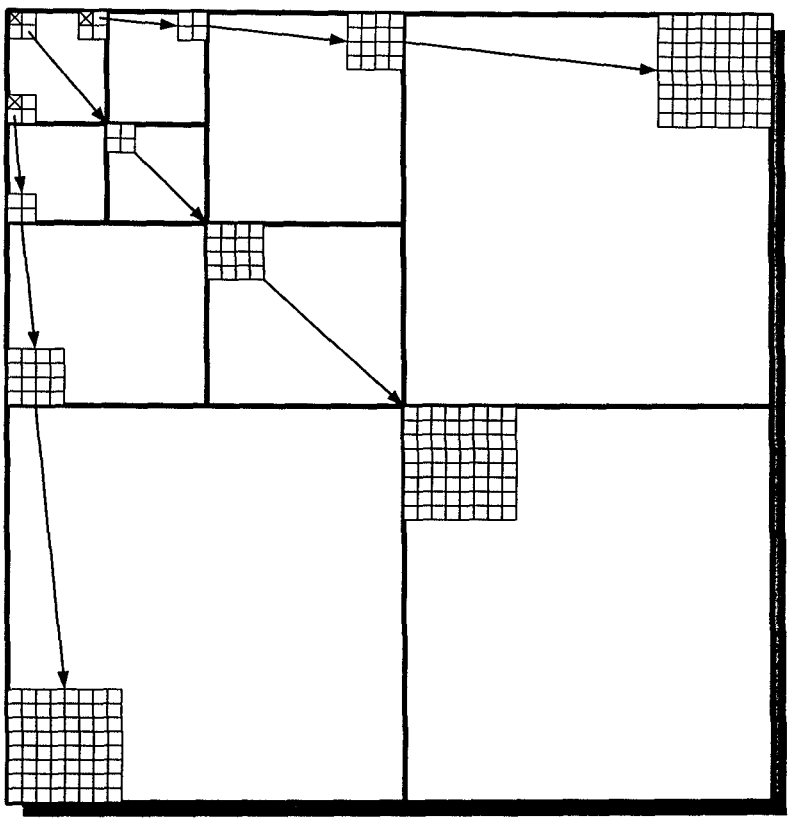
ภาพที่ 3.1.b ขั้นตอนการถอดรหัส

ภาพที่ 3.1 แผนภาพแสดงขั้นตอนการเข้ารหัสและขั้นตอนการถอดรหัส

3.2.1 ความสัมพันธ์แบบ Spatial Orientation Trees

โดยปกติพลังงานของภาพ (image energy) ที่ทำการแปลงเวฟเล็ตแบบดีสครีท พลังงานส่วนใหญ่จะอยู่ในส่วนความถี่ต่ำ หรือส่วนภาพของสัมประสิทธิ์แบนด์ย่อยความถี่ต่ำคือ $f_{LL}(x,y)$ ดังนั้นสามารถคาดหวังได้ว่าขนาดของสัมประสิทธิ์จากแบนด์ย่อยระดับการแปลงที่สูงไปยังระดับแบนด์ย่อยระดับการแปลงที่ต่ำกว่าของการแปลงเวฟเล็ต ตามทิศทางของ Spatial ตาม

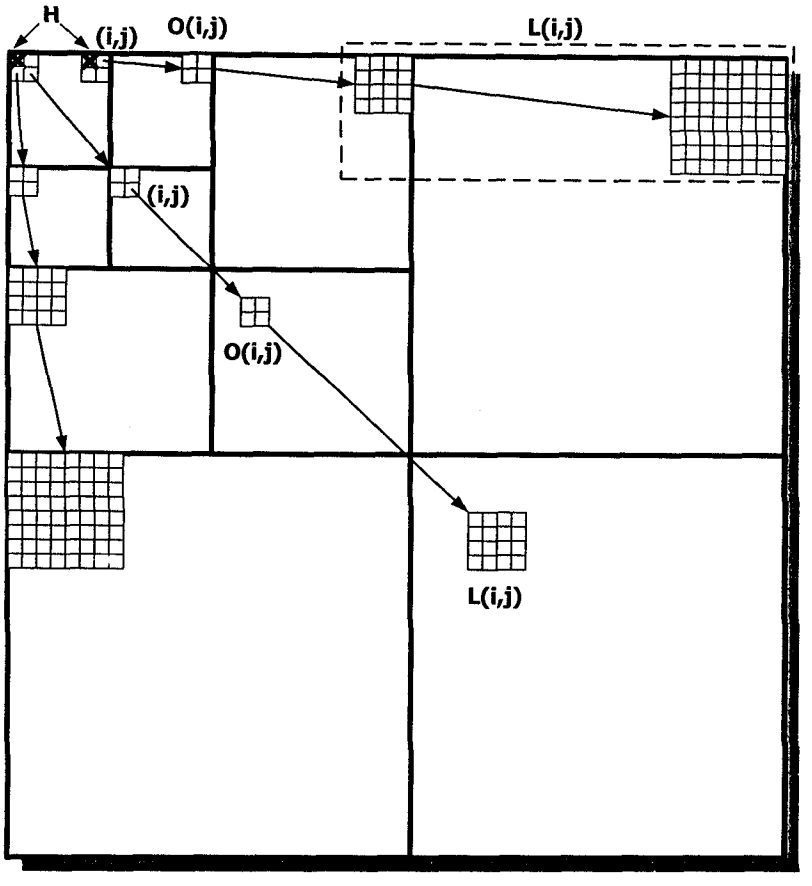
ความสัมพันธ์แบบ Spatial Orientation Trees เป็นดังภาพที่ 3.2 ขนาดของสัมประสิทธิ์จะมีค่ามากขึ้น (Amir and William, 1993)



ภาพที่ 3.2 แผนภาพแสดงความสัมพันธ์แบบ Spatial Orientation Trees

การกำหนดตำแหน่ง (coordinates) ของการเข้ารหัส ตามความสัมพันธ์แบบ spatial orientation trees เป็นดังภาพที่ 3.3 และมีตำแหน่งที่ต้องกำหนดดังนี้

- (i,j) คือ ตำแหน่งของสัมประสิทธิ์หรือโหนด (node)
- O(i,j) คือ ตำแหน่งของการสืบทอด (offspring) ของโหนด และ 1 โหนดมีการสืบทอด 4 ตำแหน่ง
- L(i,j) คือ ตำแหน่งของการสืบทอดของ O(i,j) โดยมีการสืบทอดไปจนกว่าจะถึงแบนด์ย่อยต่ำสุด
- D(i,j) = O(i,j) + L(i,j) คือ ตำแหน่งของการสืบทอดทั้งหมดของโหนด
- H คือ ตำแหน่งทั้งหมดที่อยู่ในแบนด์ย่อยสูงสุด ที่เป็นต้นกำเนิดโหนด และแต่ละตำแหน่งจะมีตำแหน่งของการสืบทอด 3 ตำแหน่ง



ภาพที่ 3.3 แผนภาพกำหนดตำแหน่งของการเข้ารหัส

3.2.2 อัลกอริทึมการเข้ารหัส SPIHT

ในการเข้ารหัสและถอดรหัส จะมีตารางในการเข้ารหัสอยู่ด้วยกัน 3 ตาราง คือ

1. List of Insignificant Set (LIS) คือ ตารางที่ใช้เก็บตำแหน่งของกลุ่มสัมประสิทธิ์เวฟเล็ตที่ยังไม่มีความสำคัญ ซึ่ง LIS นี้จะมีอยู่ด้วยกัน 2 ชนิดคือ ชนิด A และ ชนิด B ซึ่งค่าที่เก็บใน LIS ชนิด A คือ D(i,j) และค่าที่เก็บใน LIS ชนิด B คือ L(i,j)

2. List of Insignificant Pixel (LIP) คือตารางที่ใช้เก็บตำแหน่งของสัมประสิทธิ์เวฟเล็ตที่ยังไม่มีความสำคัญ ซึ่งค่าที่เก็บคือ (i,j) ที่ไม่มีความสำคัญ

3. List of Significant Pixel (LSP) คือ ตารางที่ใช้เก็บตำแหน่งของสัมประสิทธิ์เวฟเล็ตที่มีความสำคัญ ซึ่งค่าที่เก็บคือ (i,j) ที่มีความสำคัญ

มีฟังก์ชันการเข้ารหัส $S_T(k)$ เป็นดังสมการ

$$S_T(k) = \begin{cases} 1, & \max_{(i,j)\in k}|c(i,j)| \geq T \\ 0, & \text{อื่นๆ} \end{cases} \quad (3.1)$$

$$n \leq \lfloor \log_2(\max|c(i,j)|) \rfloor \quad (3.2)$$

$$T = 2^n \quad (3.3)$$

เมื่อ k คือ สัมประสิทธิ์ (c(i,j)) หรือกลุ่มของสัมประสิทธิ์ของเวฟเล็ต (D(i,j) หรือ L(i,j))

การเข้ารหัสของอัลกอริทึม SPIHT มีขั้นตอนการเข้ารหัสอยู่ด้วยกัน 4 ขั้นตอนดังนี้

ขั้นตอนที่ 1 กำหนดค่าเริ่มต้น

- หาค่า n และ T ตามสมการที่ 3.2 และสมการที่ 3.3 ตามลำดับ
- LSP กำหนดเป็นตารางว่าง
- LIP กำหนดเป็นค่าสมาชิกของ H
- LIS กำหนดเป็นค่าสมาชิกของ H และกำหนดให้เป็นชนิด A

ขั้นตอนที่ 2 การจัดลำดับ (sorting) มีขั้นตอนตามภาพที่ 3.4

- สำหรับแต่ละ (i,j) ใน LIP
  - ส่งค่า $S_T(c(i,j))$ ออก
  - ถ้าบิตเท่ากับ 1 ให้ส่งบิตเช็คเครื่องหมาย (กำหนดให้ถ้า c(i,j) มีเครื่องหมายเป็นบวก ให้ส่ง 1 และถ้า c(i,j) มีเครื่องหมายลบให้ส่ง 0)
- สำหรับแต่ละ (i,j) ใน LIS
  - ถ้าเป็น LIS ชนิด A
    - * ส่งค่า $S_T(D(i,j))$ ออก
    - * ถ้า $S_T(D(i,j))$ เท่ากับ 1

      สำหรับแต่ละ (k,l) ของ O(i,j)

      ~ ส่งค่า $S_T(c(k,l))$ ออก

      ~ ถ้าค่า $S_T(c(k,l))$ เท่ากับ 1 ให้ส่งบิตเช็คเครื่องหมาย และให้ส่ง (k,l) ไป LSP

      ~ ถ้าค่า $S_T(c(k,l))$ เท่ากับ 0 ให้ส่ง (k,l) ไป LIP
    - * ถ้า L(i,j) มีสมาชิก ให้ส่ง (i,j) ไปท้าย LIS แล้วกำหนดให้เป็นชนิด B
  - ถ้าเป็น LIS ชนิด B
    - * ส่งค่า $S_T(L(i,j))$ ออก

~ ถ้าค่า $S_T(L(i,j))$ เท่ากับ 1 ให้ส่งแต่ละสมาชิกของ O(i,j) ไปท้าย LIS แล้วกำหนดให้เป็นชนิด A และย้าย (i,j) ออกจาก LIS

ขั้นตอนที่ 3 Quantization

- n = n -1
- สำหรับแต่ละ (i,j) ใน LSP
  - ส่งค่าบิตที่ n ของ c(i,j) ออก

ขั้นตอนที่ 4 Update

- ทำซ้ำขั้นตอนที่ 2 ถึงขั้นตอนที่ 3 และยุติเมื่อมีการส่งข้อมูลการบีบอัดสัญญาณครบตามต้องการ

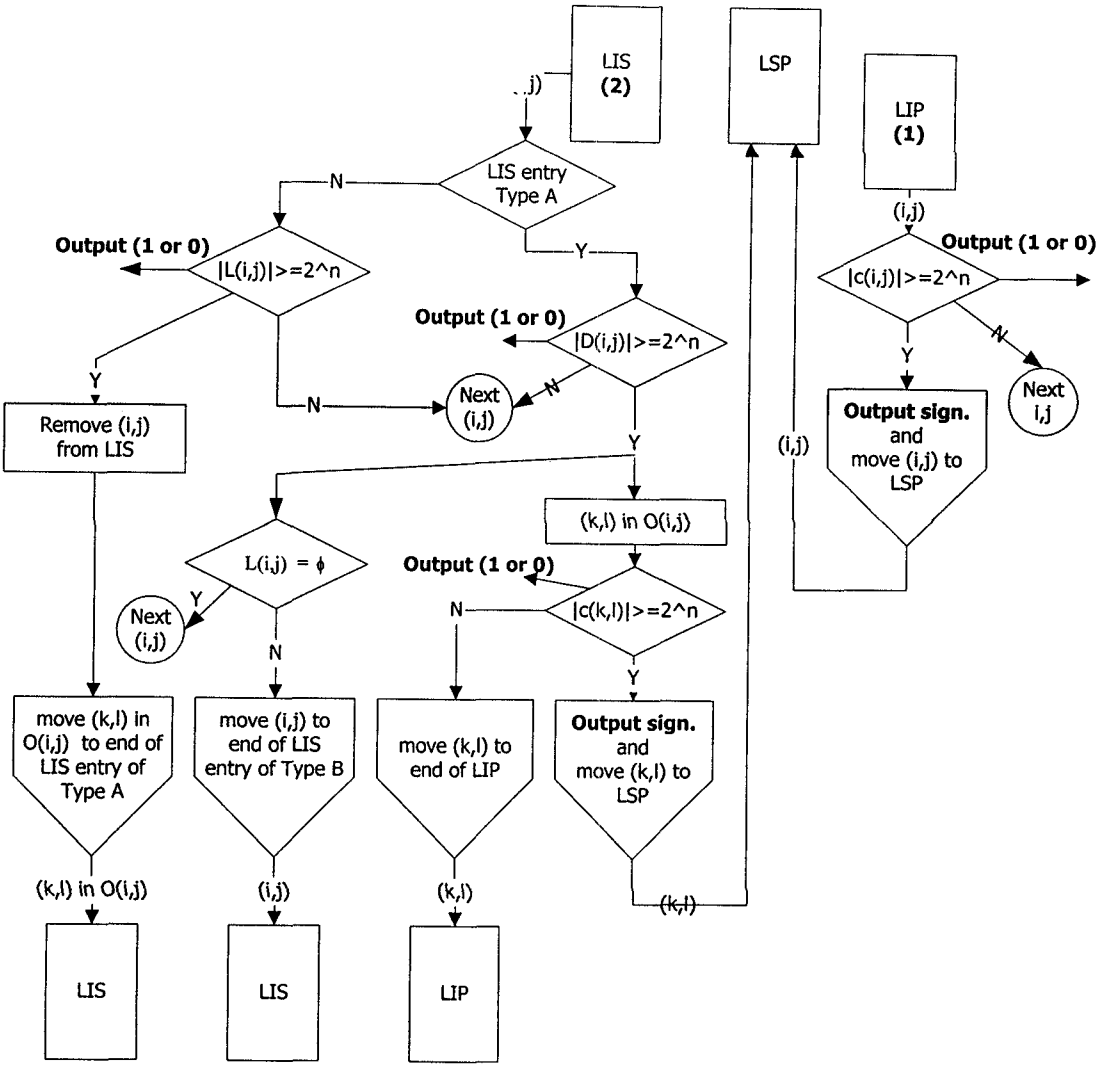
3.2.3 การจัดเก็บเป็นไฟล์การบีบอัดสัญญาณ

การจัดเก็บไฟล์แบ่งออกเป็น 2 ส่วนคือ ไฟล์ส่วนหัว และไฟล์ข้อมูลที่ได้จากการเข้ารหัส โดยไฟล์แต่ละส่วนจะมีการจัดเก็บค่าดังนี้

1. ไฟล์ส่วนหัวจะทำการจัดเก็บข้อมูลคือ
   - 1.1 จำนวนข้อมูลทั้งหมดที่ทำการบีบอัด
   - 1.2 ค่า n
   - 1.3 จำนวนคอลัมน์และจำนวนแถวของภาพ
   - 1.4 จำนวนระดับการแปลงเวฟเล็ต
2. ไฟล์ข้อมูลจะทำการจัดเก็บข้อมูลทั้งหมดที่ได้จากเข้ารหัสการบีบอัดข้อมูลด้วยวิธี SPIHT

3.2.4 อัลกอริทึมการถอดรหัส SPIHT

ในการถอดรหัสจะใช้อัลกอริทึมเดียวกับการเข้ารหัส เพียงแต่เปลี่ยนจากการส่งค่าเป็นการรับค่า และนำค่านั้นมาพิจารณาเหมือนการเข้ารหัส



ภาพที่ 3.4 แผนภาพแสดงการเข้ารหัสขั้นตอนที่ 2 ของอัลกอริทึม SPIHT

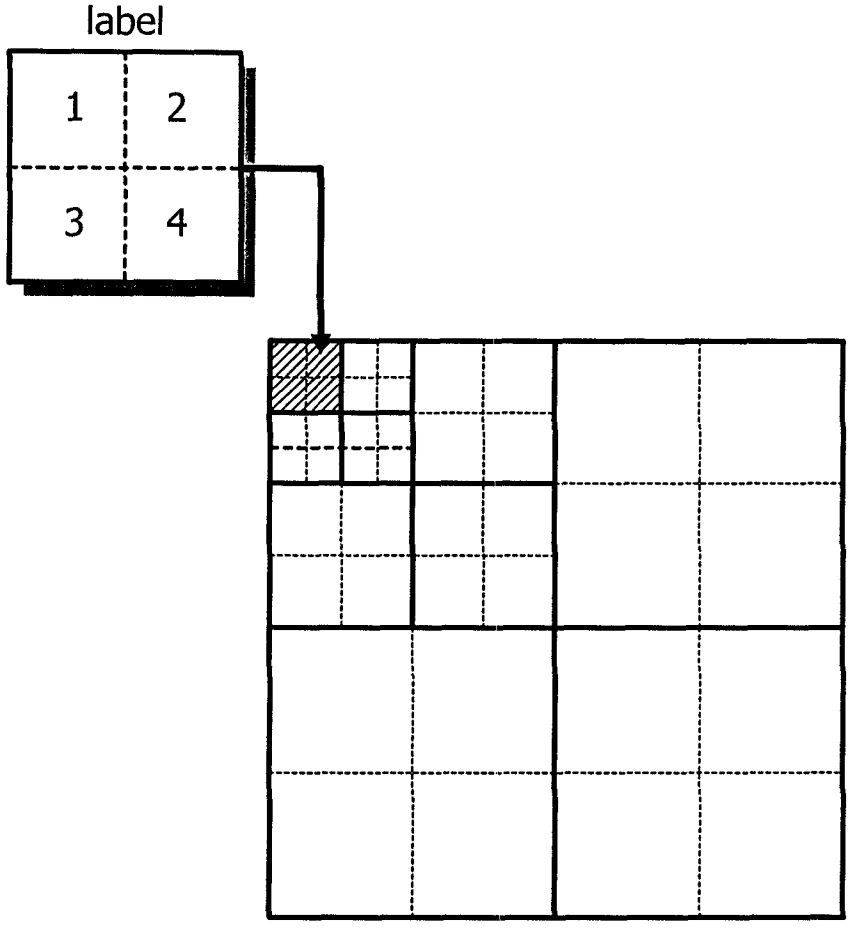
3.3 การพัฒนาอัลกอริทึม SPIHT

จากอัลกอริทึมที่กล่าวมามีจุดที่สามารถปรับปรุงพัฒนาได้ 2 จุดกล่าวคือ

กรณีที่ 1 คือ O(i,j) ไม่มีระดับความสำคัญแต่ L(i,j) มีระดับความสำคัญ จะมีการส่งบิตออกมา 2 บิต (ไม่นับรวมการเช็คแต่ละสมาชิกของ O(i,j)) ในกรณีนี้สามารถปรับอัลกอริทึมให้ส่งออกมาเพียง 1 บิต ทำได้โดยการเพิ่มการเช็คการส่ง (k,l) เข้า LIP คือถ้ามีการส่ง (k,l) ทั้งหมดของ O(i,j) แสดงว่าได้เกิดกรณีนี้ขึ้น ดังนั้นสามารถทำการส่ง (k,l) เข้า LIS ให้เป็นชนิด A ได้เลย

กรณีที่ 2 คือ O(i,j) มีระดับความสำคัญแต่ L(i,j) ไม่มีระดับความสำคัญ จะมีการส่งบิตออกมาเหมือนกับข้อ 1 ในกรณีนี้สามารถปรับอัลกอริทึมให้ส่งข้อมูลให้น้อยลงได้ ทำได้โดยการเพิ่ม List of Forbidden Coefficients (LFC) และมีขั้นตอนการหาดังนี้

- ทำการแบ่งสัมประสิทธิ์เวฟเล็ตแต่ละแบนด์ย่อยออกเป็น 4 ส่วน โดยเรียกแต่ละส่วนที่ทำการแบ่งว่า label โดยแต่ละแบนด์ย่อยที่มีการแบ่งจะมีค่าตำแหน่ง label ไม่ซ้ำกัน เป็นดังภาพที่ 3.5



ภาพที่ 3.5 แผนภาพแสดงการแบ่งแบนด์ย่อยและกำหนด label ของ LFC

- หาค่าสัมประสิทธิ์สูงสุดของแต่ละ label
- หาค่าระดับของการควอนไทซ์ในแต่ละ label และนำมาทำเป็น LFC โดยหาได้จากสมการ 3.4

$$LFC(w) \le \lfloor \log_2(\max(c_{label}(w))) \rfloor \qquad (3.4)$$

เมื่อ W คือ ตำแหน่งของ label

$C_{label}(W)$ คือ สัมประสิทธิ์เวฟเล็ตทั้งหมดที่ W

LFC(W) คือ ระดับการควอนไทซ์ ที่ W

โดย LFC นี้จะเป็นตัวบ่งบอกสถานะว่า O(i,j) และ L(i,j) ควรมีการส่งบิตออกมาหรือไม่ โดยจะดูว่า O(i,j) หรือ L(i,j) นี้อยู่ที่ label ไหน และมีระดับการควอนไทซ์ที่เท่าใด ดังนั้นถ้าระดับของการควอนไทซ์ที่มีอยู่มีค่าน้อยกว่าระดับการควอนไทซ์ของรอบการทำงาน ก็ไม่ต้องทำการเช็ค O(i,j) หรือ L(i,j) สามารถเขียนเป็นฟังก์ชันได้ดังสมการที่ 3.5

$$F_n(p) = \begin{cases} 1, & \max_{w \in p} LFC(w) \le n \\ 0, & \text{อื่นๆ} \end{cases} \qquad (3.5)$$

เมื่อ $F_n(p)$ คือ ฟังก์ชันที่ใช้เป็นเงื่อนไขในการเช็คค่าระดับการควอนไทซ์ของ O(i,j) หรือ L(i,j)

p คือ ค่า LFC ทั้งหมดที่เป็นของสมาชิกใน O(i,j) หรือ L(i,j)

3.3.1 อัลกอริทึมการเข้ารหัสอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

การเข้ารหัสของอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง มีขั้นตอนการเข้ารหัสอยู่ด้วยกัน 4 ขั้นตอนดังนี้

ขั้นตอนที่ 1 กำหนดค่าเริ่มต้น

- หาค่า n และ T ตามสมการที่ 3.2 และสมการที่ 3.3 ตามลำดับ
- LSP กำหนดเป็นตารางว่าง
- LIP กำหนดเป็นค่าสมาชิกของ H
- LIS กำหนดเป็นค่าสมาชิกของ H และกำหนดให้เป็นชนิด A
- LFC กำหนดค่าเป็นไปตามสมการที่ 3.4

ขั้นตอนที่ 2 การจัดลำดับ มีขั้นตอนตามภาพที่ 3.6

- สำหรับแต่ละ (i,j) ใน LIP
  - ส่งค่า $S_T(c(i,j))$ ออก
  - ถ้าบิตเท่ากับ 1 ให้ส่งบิตเช็คเครื่องหมาย
- สำหรับแต่ละ (i,j) ใน LIS
  - เช็คค่า FO จาก $F_n(O(i,j))$
  - เช็คค่า FL จาก $F_n(L(i,j))$
  - ถ้าเป็น LIS ชนิด A และ FO หรือ FL เท่ากับ 1
    - * ส่งค่า $S_T(D(i,j))$ ออก
    - * ถ้า $S_T(D(i,j))$ เท่ากับ 1

      สำหรับแต่ละ (k,l) ของ O(i,j)

~ ส่งค่า $S_T(c(k,l))$ ออก

~ ถ้าค่า $S_T(c(k,l))$ เท่ากับ 1 ให้ส่งบิตเช็คเครื่องหมาย และให้ส่ง (k,l) ไป LSP

~ ถ้าค่า $S_T(c(k,l))$ เท่ากับ 0 ให้ส่ง (k,l) ไป LIP

* ถ้า L(i,j) มีสมาชิก

~ ถ้าสมาชิกทุกตัวของ O(i,j) ถูกส่งไป LIP ให้ทำการส่งสมาชิกทุกตัวของ O(i,j) ไปท้าย LIS และกำหนดให้เป็นชนิด A

~ ถ้าสมาชิกบางตัวของ O(i,j) ไม่ถูกส่งไป LIP ให้ทำการส่ง (i,j) ไปท้าย LIS แล้วกำหนดให้เป็นชนิด B

- ถ้าเป็น LIS ชนิด B และ FL เท่ากับ 1

* ส่งค่า $S_T(L(i,j))$ ออก

~ ถ้าค่า $S_T(L(i,j))$ เท่ากับ 1 ให้ส่งแต่ละสมาชิกของ O(i,j) ไปท้าย LIS แล้วกำหนดให้เป็นชนิด A และทำการย้าย (i,j) ออกจาก LIS

ขั้นตอนที่ 3 Quantization

- n = n -1
- สำหรับแต่ละ (i,j) ใน LSP
  - ส่งค่าบิตที่ n ของ c(i,j) ออก

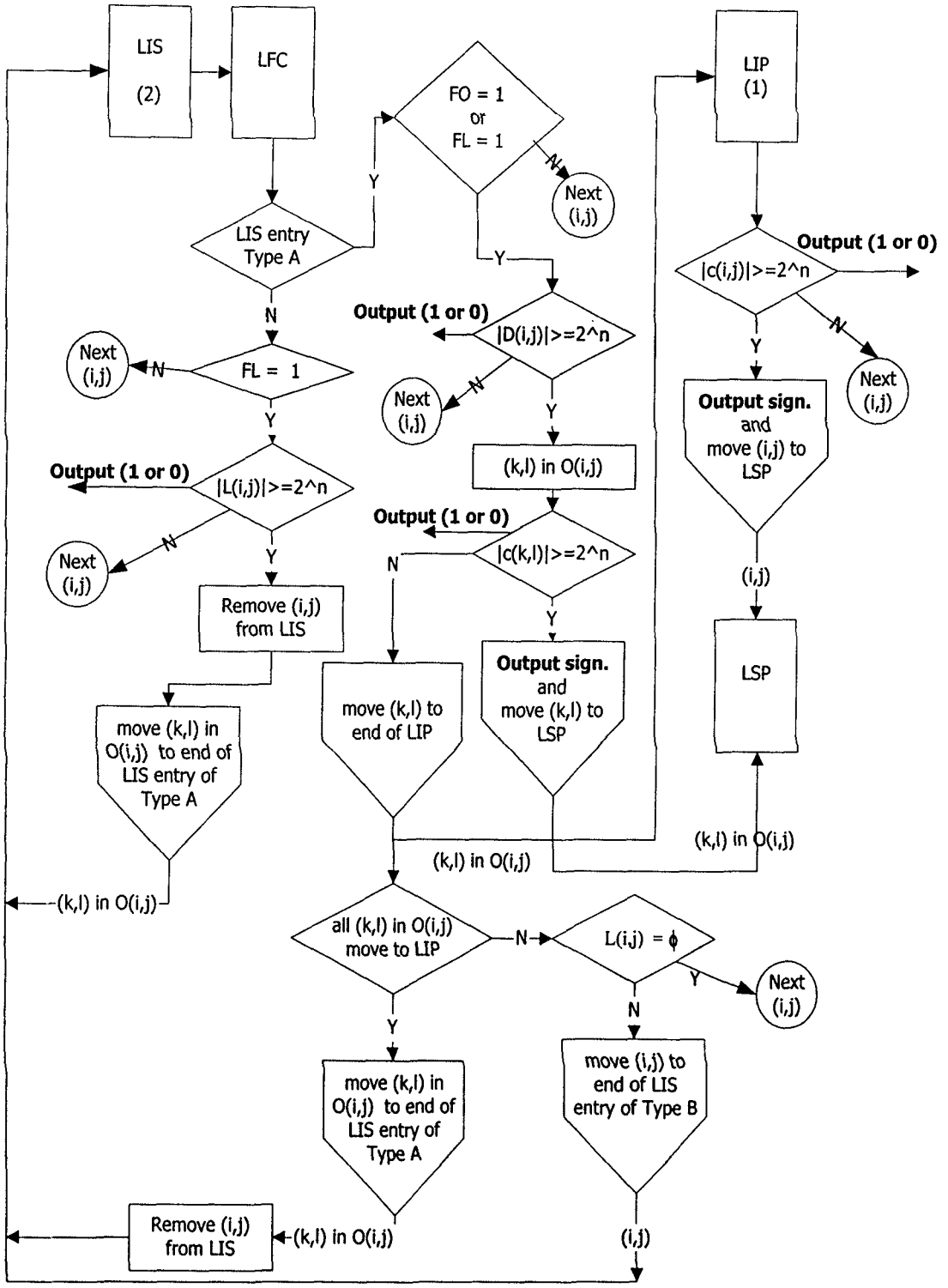
ขั้นตอนที่ 4 Update

- ทำซ้ำขั้นตอนที่ 2 ถึงขั้นตอนที่ 3 และยุติเมื่อมีการส่งข้อมูลการบีบอัดสัญญาณครบตามต้องการ

3.3.2 การจัดเก็บเป็นไฟล์การบีบอัดข้อมูลอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

การจัดเก็บไฟล์แบ่งออกเป็น 2 ส่วนคือ ไฟล์ส่วนหัว และไฟล์ข้อมูลที่ได้จากการเข้ารหัส โดยไฟล์แต่ละส่วนจะมีการจัดเก็บค่าดังนี้

1. ไฟล์ส่วนหัวจะทำการจัดเก็บข้อมูลคือ
   1.1 จำนวนข้อมูลทั้งหมดที่ทำการบีบอัด
   1.2 ค่า n
   1.3 จำนวนคอลัมน์และจำนวนแถวของภาพ
   1.4 จำนวนระดับการแปลงเวฟเล็ต
   1.5 ค่า LFC
2. ไฟล์ข้อมูลจะทำการจัดเก็บข้อมูลทั้งหมดที่ได้จากเข้ารหัสการบีบอัดข้อมูลด้วยวิธี SPIHT ที่ผ่านการปรับปรุง



ภาพที่ 3.6 แผนภาพแสดงการเข้ารหัสขั้นตอนที่ 2 ของอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

3.3.3 อัลกอริทึมการถอดรหัสอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

ในการถอดรหัสจะใช้อัลกอริทึมเดียวกับการเข้ารหัส เพียงแต่เปลี่ยนจากการส่งค่าเป็นการรับค่า และนำค่านั้นมาพิจารณาเหมือนการเข้ารหัส

3.5 สรุป

ในบทนี้กล่าวถึงการเข้ารหัสและถอดรหัสด้วยอัลกอริทึม SPIHT และกล่าวถึงการพัฒนาอัลกอริทึม SPIHT ด้วยการเพิ่มเงื่อนไขการเช็คการส่ง (k,l) เข้า LIP คือถ้ามีการส่ง (k,l) ทั้งหมดของ O(i,j) แสดงว่าได้เกิดกรณีนี้ขึ้น ดังนั้นสามารถทำการส่ง (k,l) เข้า LIS ให้เป็นชนิด A ได้เลย เพื่อลดการส่งข้อมูลในเงื่อนไข O(i,j) ไม่มีระดับความสำคัญแต่ L(i,j) มีระดับความสำคัญ และทำการเพิ่ม LFC และหาค่า FO และ FL เพื่อนำไปตัดสินใจว่าจะทำการลดการส่งข้อมูลในเงื่อนไข O(i,j) มีระดับความสำคัญแต่ L(i,j) ไม่มีระดับความสำคัญหรือไม่ นอกจากนั้นยังกล่าวถึงการนำการเข้ารหัสเลขคณิตมาใช้กับข้อมูลที่ได้จากอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง เพื่อทำการเพิ่มประสิทธิภาพการบีบอัดข้อมูลให้สูงขึ้น

บทที่ 4

ผลการทดลอง

4.1 บทนำ

การบีบอัดสัญญาณภาพถ่ายจากดาวเทียมด้วยการแปลงเวฟเล็ตแบบดีสครีท แล้วทำการเข้ารหัสด้วยอัลกอริทึม SPIHT และทำการเพิ่มประสิทธิภาพการบีบอัดสัญญาณ ด้วยการนำข้อมูลที่ได้จากอัลกอริทึม SPIHT ไปทำการเข้ารหัสเลขคณิต ในการแปลงเวฟเล็ตถ้ามีการเปลี่ยนเวฟเล็ตแม่ ก็จะทำให้ค่าสัมประสิทธิ์เวฟเล็ตที่ได้แตกต่างกันไป ซึ่งก็จะทำให้ประสิทธิภาพการบีบอัดสัญญาณภาพแตกต่างกันไปด้วย ดังนั้นจึงต้องมีการทดสอบหาเวฟเล็ตแม่ ที่ทำให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง ให้ได้ผลการบีบอัดที่ดีที่สุด

4.2 การวัดประสิทธิภาพการบีบอัดสัญญาณภาพ

ในการบีบอัดสัญญาณภาพแบบมีการสูญเสีย ประสิทธิภาพของเทคนิคต่างๆ ไม่สามารถสรุปได้จากอัตราการบีบอัดสัญญาณเพียงอย่างเดียว แต่ต้องพิจารณาถึงความสูญเสียสัญญาณที่เกิดขึ้นว่ายอมรับได้หรือไม่ เกณฑ์ที่ใช้ในการวัดประสิทธิภาพของการบีบอัดสัญญาณภาพสามารถแบ่งออกเป็น 2 ประเภทดังนี้

4.2.1 การวัดเชิงปริมาณ

การวัดค่าความผิดพลาดที่เกิดจากการบีบอัดสัญญาณภาพสามารถกระทำได้หลายวิธีการคือ

4.2.1.1 ค่าความผิดพลาดกำลังสองเฉลี่ย (mean square error: MSE)

ค่าความผิดพลาดกำลังสองเฉลี่ยของการบีบอัดสัญญาณภาพขนาด MxN หาได้ดังนี้

$$MSE = \frac{1}{M * N} \sum_{x=1}^{M} \sum_{y=1}^{N} \left[f(x,y) - f_i(x,y) \right]^2 \qquad (4.1)$$

โดยที่ M คือจำนวนพิกเซลตามความกว้างของภาพ

N คือจำนวนพิกเซลตามความสูงของภาพ

$f(x,y)$ คือค่าของพิกเซลที่ตำแหน่ง (x,y) ของภาพต้นแบบ

$f_i(x,y)$ คือค่าของพิกเซลที่ตำแหน่ง (x,y) ของภาพที่สร้างกลับคืนมาใหม่

ค่าความผิดพลาดกำลังสองเฉลี่ยถ้ามีค่าน้อย แสดงว่าภาพที่ได้จากการสร้างกลับจากการบีบอัดสัญญาณมีความผิดเพี้ยนไปจากภาพต้นแบบน้อย และถ้าค่าความผิดพลาดกำลังสองเฉลี่ยมีค่ามาก แสดงว่าภาพที่ได้จากการสร้างกลับจากการบีบอัดสัญญาณมีความผิดเพี้ยนไปจากภาพต้นแบบมาก

4.2.1.2 อัตราส่วนของสัญญาณต่อสัญญาณรบกวนสูงสุด (peak-signal-to-noise ratio: PSNR)

ค่าอัตราส่วนของสัญญาณต่อสัญญาณรบกวนสูงสุดของการบีบอัดสัญญาณภาพขนาด MxN ของภาพระดับความเทา (gray scale) หาได้ดังนี้

$$PSNR = 10 \log \frac{255^2}{MSE} \qquad (dB) \tag{4.2}$$

ค่าอัตราส่วนของสัญญาณต่อสัญญาณรบกวนสูงสุดถ้ามีค่าน้อย แสดงว่าภาพที่ได้จากการสร้างกลับจากการบีบอัดสัญญาณมีความผิดเพี้ยนไปจากภาพต้นแบบมาก และถ้าอัตราส่วนของสัญญาณต่อสัญญาณรบกวนสูงสุดมีค่ามาก แสดงว่าภาพที่ได้จากการสร้างกลับจากการบีบอัดสัญญาณมีความผิดเพี้ยนไปจากภาพต้นแบบน้อย

4.2.1.3 อัตราบิต (bit rate)

อัตราบิตคือค่าเฉลี่ยของจำนวนบิตต่อพิกเซล (bits per pixel: bpp) ของภาพที่ถูกลดสัญญาณ โดยสามารถคำนวณได้จากอัตราส่วนระหว่างจำนวนบิตของภาพที่ผ่านการบีบอัดสัญญาณทั้งหมดต่อจำนวนพิกเซลของภาพต้นแบบ หาได้ดังนี้

$$\text{Bit rate} = \frac{\text{จำนวนบิตทั้งหมดของภาพที่ผ่านการบีบอัดข้อมูล}}{\text{จำนวนพิกเซลทั้งหมดของภาพต้นแบบ}} \tag{4.3}$$

4.2.1.4 อัตราการบีบอัดสัญญาณภาพ (compression ration: CR)

อัตราการบีบอัดสัญญาณภาพคืออัตราส่วนระหว่างจำนวนบิตสัญญาณภาพต้นแบบและจำนวนบิตสัญญาณของภาพที่ผ่านการบีบอัดสัญญาณภาพ หาได้ดังนี้

$$CR = \frac{\text{จำนวนบิตทั้งหมดของภาพต้นแบบ}}{\text{จำนวนบิตทั้งหมดของภาพที่ผ่านการบีบสัญญาณ}} \tag{4.4}$$

4.2.2 การวัดเชิงคุณภาพ

เป็นวิธีการพื้นฐานแต่ให้ผลที่น่าเชื่อถือมาก โดยใช้สายตาของผู้ที่ไม่เกี่ยวข้องทำการตัดสินคุณภาพของภาพในฐานะที่เป็นผู้ใช้ภาพในการวิจัย และผู้ที่ใช้เครื่องมือในการถ่ายภาพ ที่มีความสามารถในการสังเกตความบกพร่องเล็กน้อยที่ผู้ใช้ทั่วไปอาจมองข้าม

4.3 การหาตระกูลเวฟเล็ตแม่

ในขั้นตอนการแปลงเวฟเล็ตการใช้เวฟเล็ตแม่ที่ต่างกัน ย่อมทำให้ได้สัมประสิทธิ์เวฟเล็ตที่ไม่เหมือนกัน ซึ่งทำให้ได้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมที่อัตราการบีบอัดสัญญาณเดียวกันแตกต่างกันออกไป ดังนั้นเพื่อให้ได้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมที่ดีที่สุด ด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง จึงต้องทำการทดสอบหาตระกูลเวฟเล็ตแม่ที่เหมาะสมสำหรับภาพถ่ายจากดาวเทียม

ตระกูลเวฟเล็ตแม่ที่ใช้ในการทดสอบ คือ Biorthogonal9-7 (bi9-7), Daubechies4 (db4), Symlets8 (sym8) และ Coiflets5 (coif5) โดยแต่ละตระกูลเวฟเล็ตแม่มีค่าดังนี้

ตระกูล bi9-7 ตัวฟิลเตอร์ $h_0(n)$, $h_1(n)$, $g_0(n)$ และ $g_1(n)$ มีความสัมพันธ์กันดังนี้ (Jabran, 2001)

$$g_0(n) = (-1)^{(n+1)} h_1(n) \tag{4.5}$$

$$g_1(n) = (-1)^{(n)} h_0(n) \tag{4.6}$$

ฟิลเตอร์ $h_0(n)$ ตระกูล bi9-7 (Marc et al., 1992) คือ

[0.03782879857992, -0.02384929751586, -0.11062402748951, 0.37740268810913,
0.85269865321930, 0.37740268810913, -0.11062402748951, -0.02384929751586,
0.03782879857992]

ฟิลเตอร์ $h_1(n)$ ตระกูล bi9-7 (Marc et al., 1992) คือ

[0.06453905013246, -0.04068975261660, -0.41809244072573, 0.7884848722061,
-0.41809244072573, -0.04068975261660, 0.06453905013246]

ตระกูล db4, sym8 และ coif5 ตัวฟิลเตอร์ $h_0(n)$, $h_1(n)$, $g_0(n)$ และ $g_1(n)$ มีความสัมพันธ์กันดังนี้ (Jabran, 2001)

$$g_0(n) = h_0(-n) \tag{4.7}$$

$$g_1(n) = (-1)^{(n)} h_0(n) \tag{4.8}$$

$$h_1(n) = (-1)^{(n+1)} h_0(-n) \tag{4.9}$$

ฟิลเตอร์ $h_0(n)$ ตระกูล db4 (Michel et al., 1996) คือ

[-0.01059740178500, 0.03288301166698, 0.03084138183599, -0.18703481171888,
-0.02798376941698, 0.63088076792959, 0.71484657055254, 0.23037781330886]

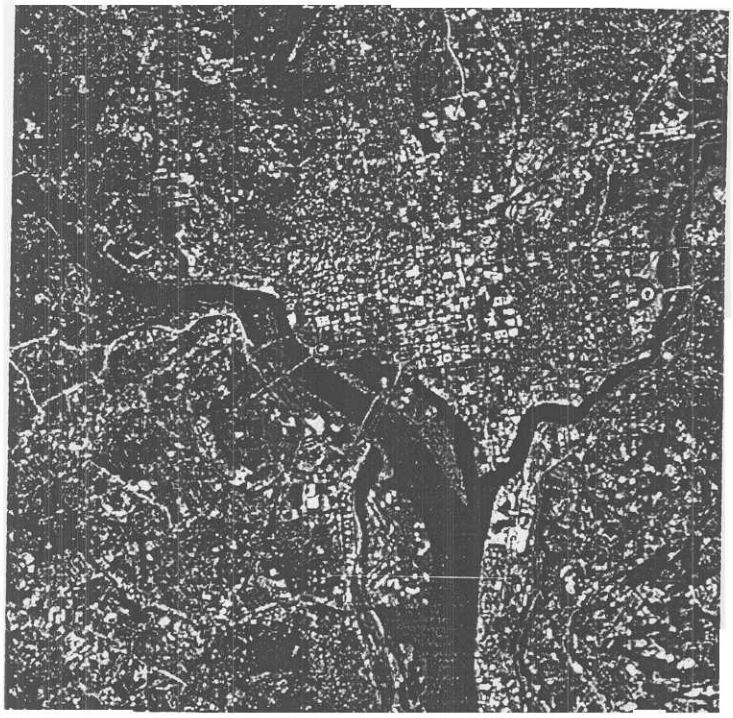
ฟิลเตอร์ $h_0(n)$ ตระกูล sym8 (Michel et al., 1996) คือ

[-0.00338241595101, -0.00054213233179, 0.03169508781149, 0.00760748732492,
-0.14329423835081, -0.06127335906766, 0.48135965125837, 0.77718575170052,
0.36444189483533, -0.05194583810771, -0.02721902991706, 0.04913717967361,
0.00380875201389, -0.01495225833705, -0.00030292051472, 0.00188995033276]

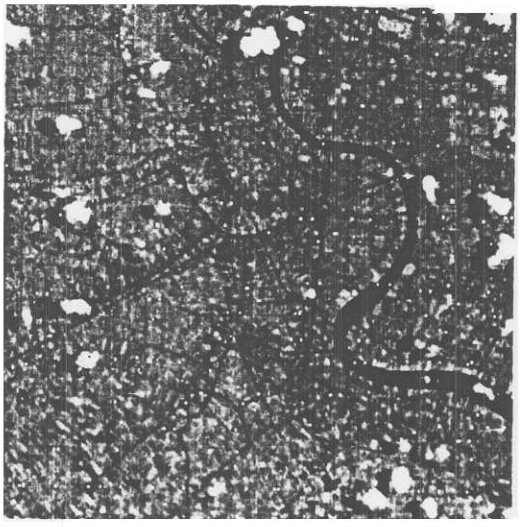
ฟิลเตอร์ $h_0(n)$ ตระกูล coif5 (Michel et al., 1996) คือ

[-0.00000009517657, -0.00000016744289, 0.00000206376185, 0.00000373465518,
-0.00002131502681, -0.00004134043227, 0.00014054114970, 0.00030225958181,
-0.00063813134305, -0.00166286370201, 0.00243337321266, 0.00676418544805,
-0.00916423116248, -0.01976177894257, 0.03268357426711, 0.04128920875018,
-0.10557420870334, -0.06203596396290, 0.43799162617184, 0.77428960365296,
0.42156620669085, -0.05204316317624, -0.09192001055970, 0.02816802897094,
0.02340815678584, -0.01013111751985, -0.00415935878139, 0.00217823635811,
0.00035858968790, -0.00021208083980]

ภาพต้นแบบที่ใช้ทดสอบคือ ภาพ sat1 ดังแสดงในภาพที่ 4.1 ซึ่งมีสัญญาณภาพขนาด 512x512 พิกเซล แต่ละพิกเซลมี 8 บิต และภาพ sat2 ดังแสดงในภาพที่ 4.2 ซึ่งมีสัญญาณภาพขนาด 256x256 พิกเซล แต่ละพิกเซลมี 8 บิต ทำการทดสอบที่อัตราบิต 0.125, 0.250, 0.500 และ 1.000 bpp และทำการวัดประสิทธิภาพด้วยวิธี PSNR



ภาพที่ 4.1 ภาพต้นแบบ (sat1)



ภาพที่ 4.2 ภาพต้นแบบ (sat2)

4.4 ผลการทดสอบการหาตระกูลเวฟเล็ตแม่

ผลการทดสอบหาตระกูลเวฟเล็ตแม่ที่ให้ผลการบีบสัญญาณภาพถ่ายจากดาวเทียมได้ดีที่สุด พบว่าทั้งอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง ที่ใช้ bi9-7 เป็นเวฟเล็ตแม่ในการแปลงเวฟเล็ต ให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมภาพ sat1 และภาพ sat2 ได้ดีที่สุด เป็นดังตารางที่ 4.1 และ ตารางที่ 4.2 ตามลำดับ

ตารางที่ 4.1 แสดงผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม
ที่ตระกูลเวฟเล็ตแม่ต่างๆ (ภาพต้นแบบ sat1)

ตระกูลเวฟเล็ตแม่	อัตราบิต (bpp)	PSNR (dB)	
		SPIHT	SPIHT*
bi9-7	0.125	19.708	19.817
	0.250	20.872	21.020
	0.500	22.788	22.901
	1.000	25.316	25.519
db4	0.125	19.577	19.645
	0.250	20.750	20.903
	0.500	22.572	22.678
	1.000	25.070	25.243
sym8	0.125	19.638	19.699
	0.250	20.839	21.005
	0.500	22.684	22.775
	1.000	25.203	25.357
coif5	0.125	19.659	19.701
	0.250	20.829	20.998
	0.500	22.687	22.769
	1.000	25.209	25.351

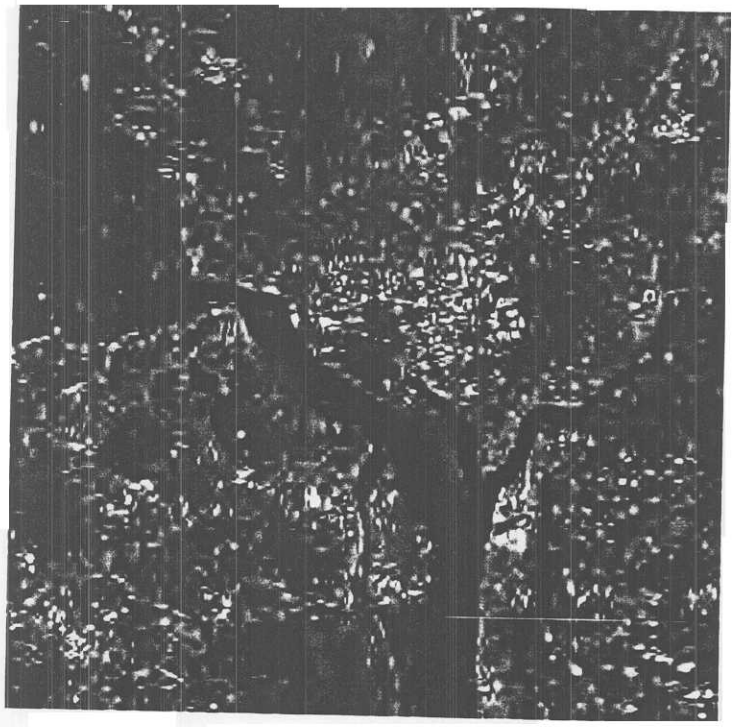
SPIHT* คือ SPIHT ที่ผ่านการปรับปรุง

ตารางที่ 4.2 แสดงผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม
ที่ตระกูลเวฟเล็ตแม่ต่างๆ (ภาพต้นแบบ sat2)

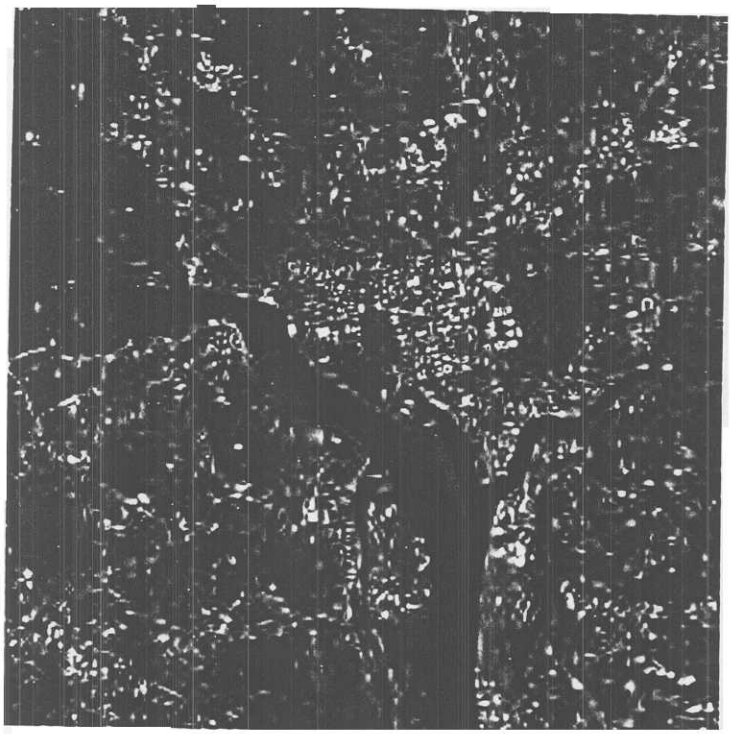
ตระกูลเวฟเล็ตแม่	อัตราบิต (bpp)	PSNR (dB)	
		SPIHT	SPIHT*
bi9-7	0.125	21.971	22.065
	0.250	23.2408	23.504
	0.500	25.174	25.476
	1.000	28.070	28.369
db4	0.125	21.712	21.856
	0.250	23.020	23.232
	0.500	25.010	25.275
	1.000	27.826	28.044
sym8	0.125	21.868	21.943
	0.250	23.142	23.372
	0.500	25.148	25.388
	1.000	28.046	28.233
coif5	0.125	21.931	21.987
	0.250	23.132	23.350
	0.500	25.135	25.361
	1.000	28.036	28.222

SPIHT* คือ SPIHT ที่ผ่านการปรับปรุง

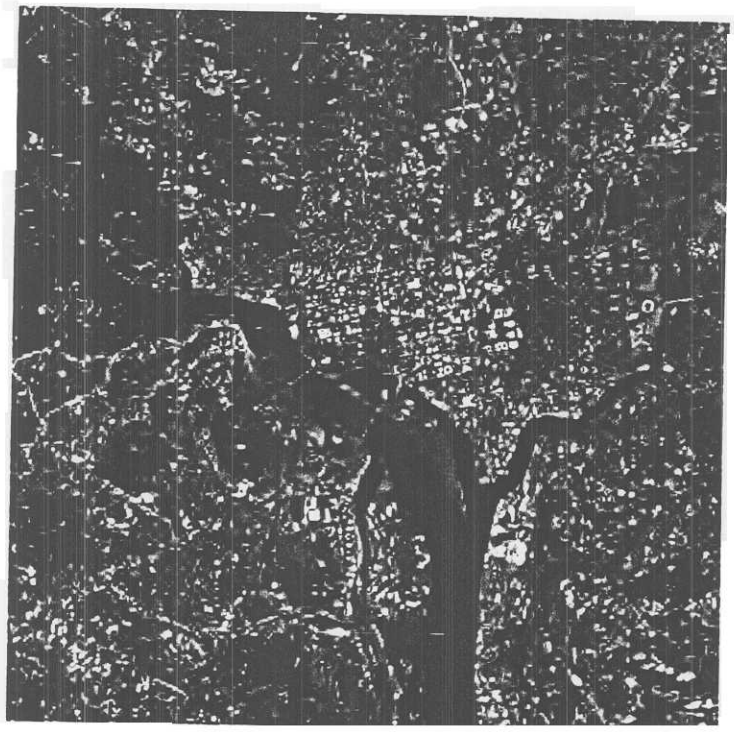
ภาพ sat1 และ sat2 ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง ที่ใช้ bi9-7 เป็นเวฟเล็ตแม่ในการแปลงเวฟเล็ตเป็นดังภาพที่ 4.3 และ4.4 ตามลำดับ



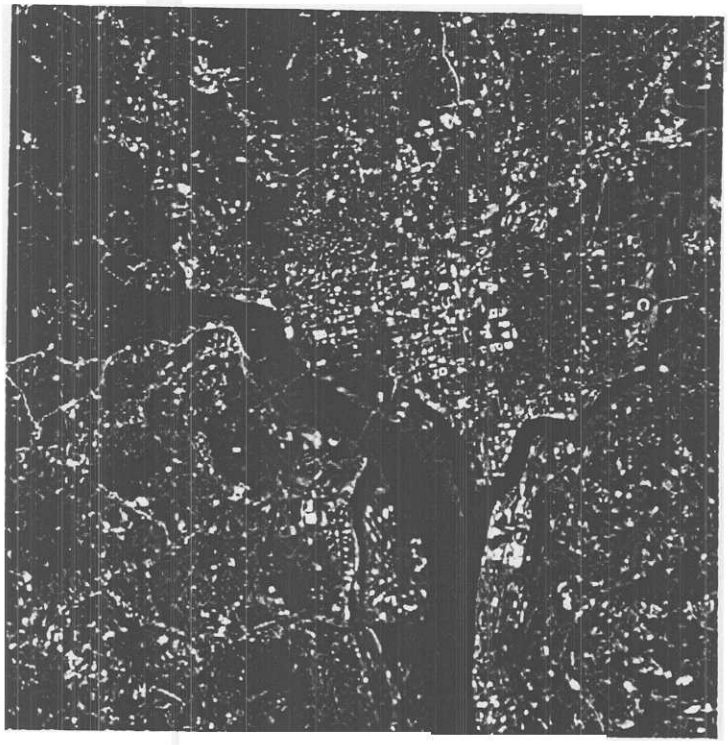
ภาพที่ 4.3.a อัตราบิต 0.125 bpp



ภาพที่ 4.3.b อัตราบิต 0.250 bpp

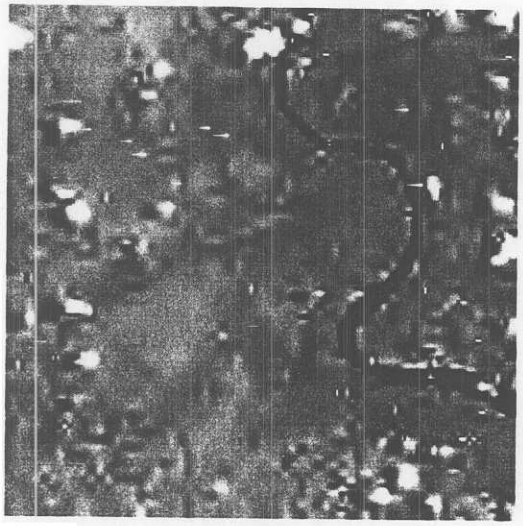


ภาพที่ 4.3.c อัตราบิต 0.500 bpp

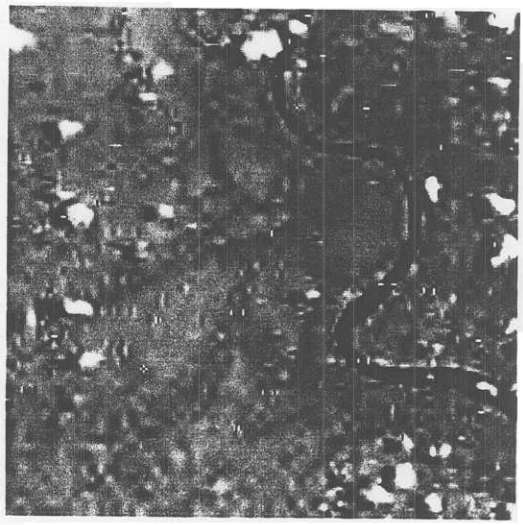


ภาพที่ 4.3.c อัตราบิต 1.000 bpp

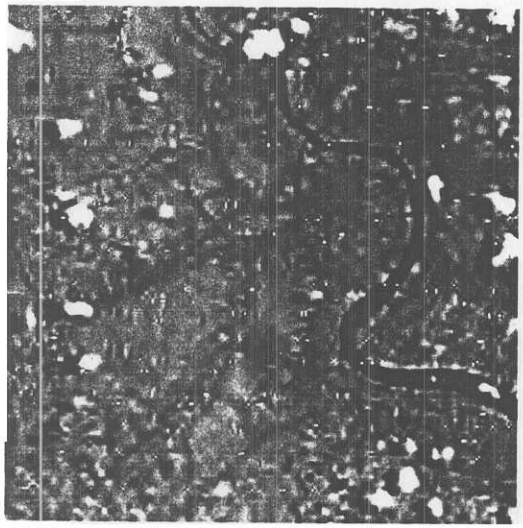
ภาที่ 4.3 ภาพ sat1 ที่ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ
ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง



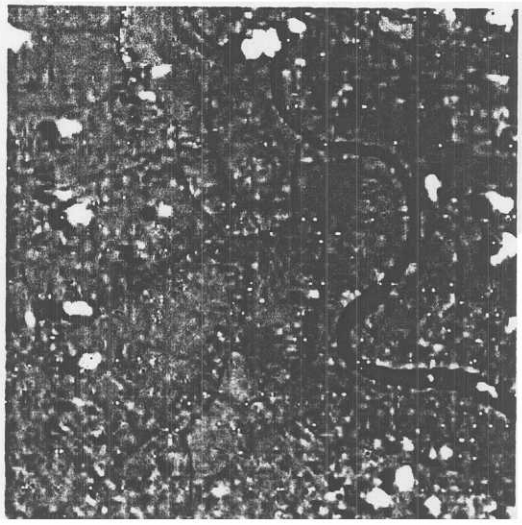
ภาพที่ 4.4.a อัตราบิต 0.125 bpp



ภาพที่ 4.4.b อัตราบิต 0.250 bpp



ภาพที่ 4.4.c อัตราบิต 0.500 bpp

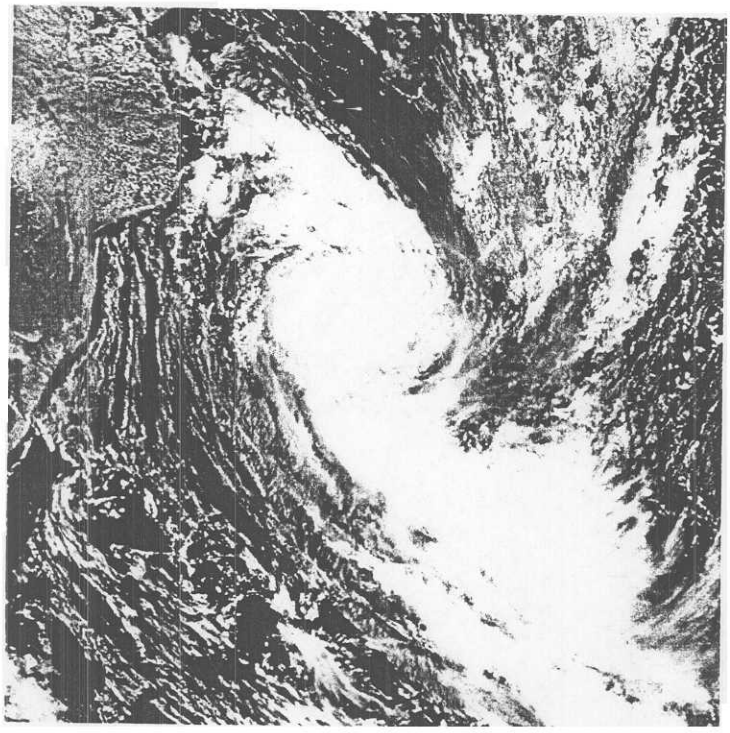


ภาพที่ 4.4.d อัตราบิต 1.000 bpp

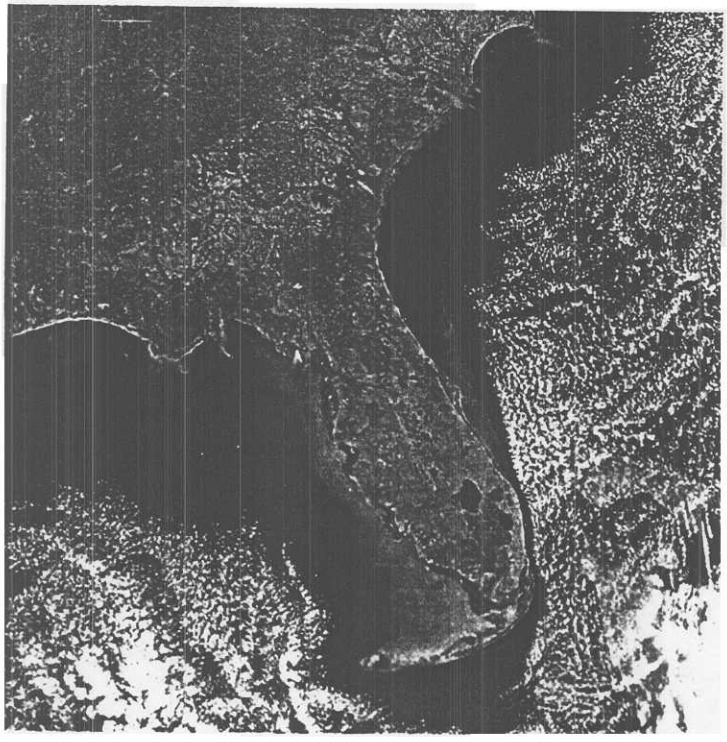
ภาที่ 4.4 ภาพ sat2 ที่ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ
ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

4.5 การบีบอัดสัญญาณภาพด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

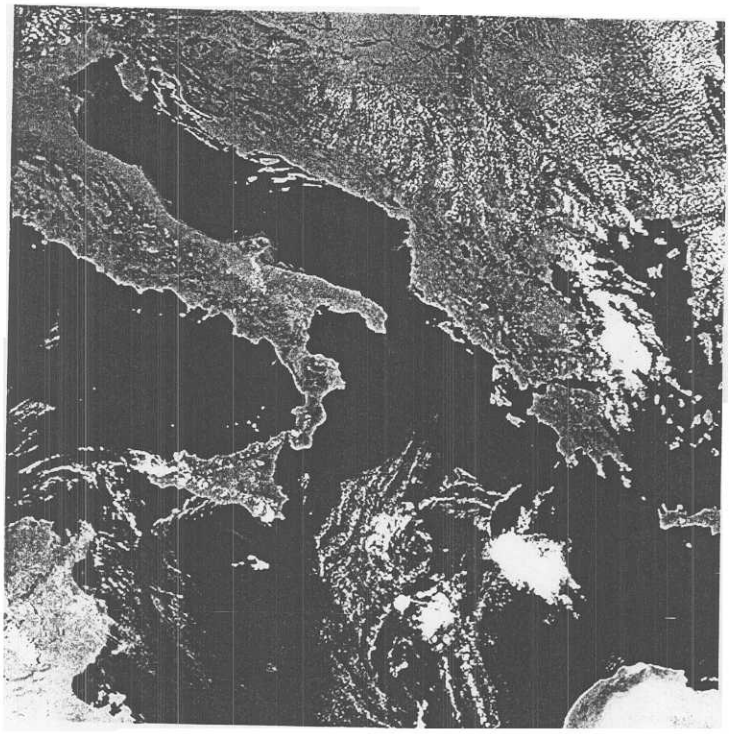
ทำการทดสอบอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง ในการแปลงเวฟเล็ตใช้ bi9-7 เป็นเวฟเล็ต ที่อัตราบิตเดียวกันคือ 0.125, 0.250, 0.500 และ 1.000 bpp โดยใช้ภาพต้นแบบจำนวน 4 ภาพ คือ ภาพ Storm, ภาพ Florida, ภาพ Mediterranean และภาพ California ซึ่งแต่ละภาพมีข้อมูลภาพขนาด 512x512 พิกเซล และแต่ละพิกเซลมี 8 บิต ซึ่งสัญญาณภาพที่นำมาทดสอบได้แสดงไว้ในภาพที่ 4.5(a-d)



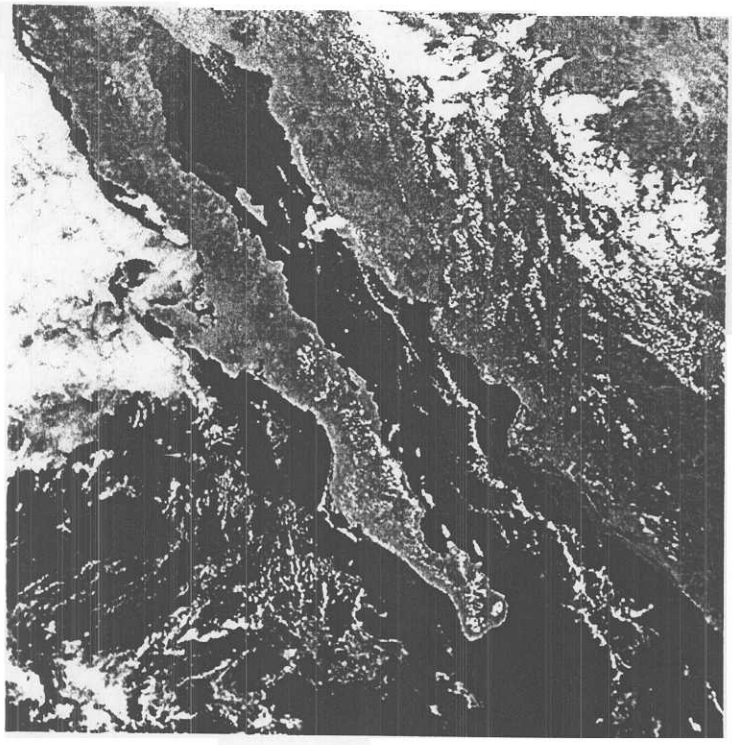
ภาพที่ 4.5.a Storm



ภาพที่ 4.5.b Florida



ภาพที่ 4.5.c Mediterranean



ภาพที่ 4.5.d California

ภาพที่ 4.5 ภาพต้นแบบที่ใช้ทดสอบการบีบอัดสัญญาณภาพดาวเทียม

4.6 ผลการทดสอบการบีบอัดสัญญาณภาพด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

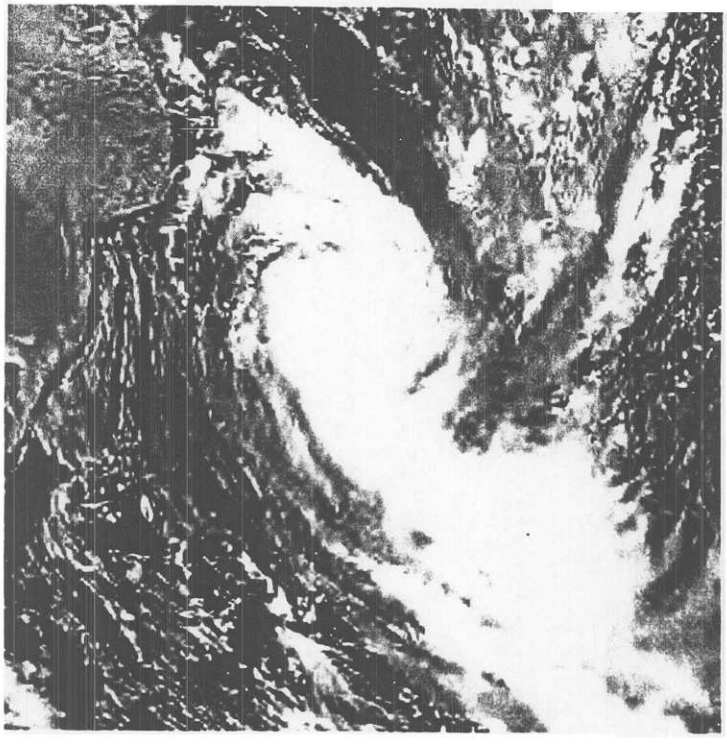
ผลการทดสอบการเปรียบเทียบประสิทธิภาพการบีบอัดสัญญาณภาพดาวเทียม ด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง ที่ใช้ bi9-7 เป็นเวฟเล็ตแม่ในการแปลงเวฟเล็ต พบว่าอัลกอริทึม SPIHT ที่ผ่านการปรับปรุงให้ผลการบีบอัดสัญญาณภาพดาวเทียม ได้ดีกว่าเป็นดังตารางที่ 4.3 และมีภาพที่ผ่านการบีบอัดสัญญาณภาพด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง เป็นดังภาพที่ 4.6 – 4.9

ตารางที่ 6.3 แสดงผลการบีบอัดข้อมูลภาพถ่ายจากดาวเทียม ด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุงปรุง ที่ใช้ bi9-7 เป็นเวฟเล็ตแม่ในการแปลงเวฟเล็ต

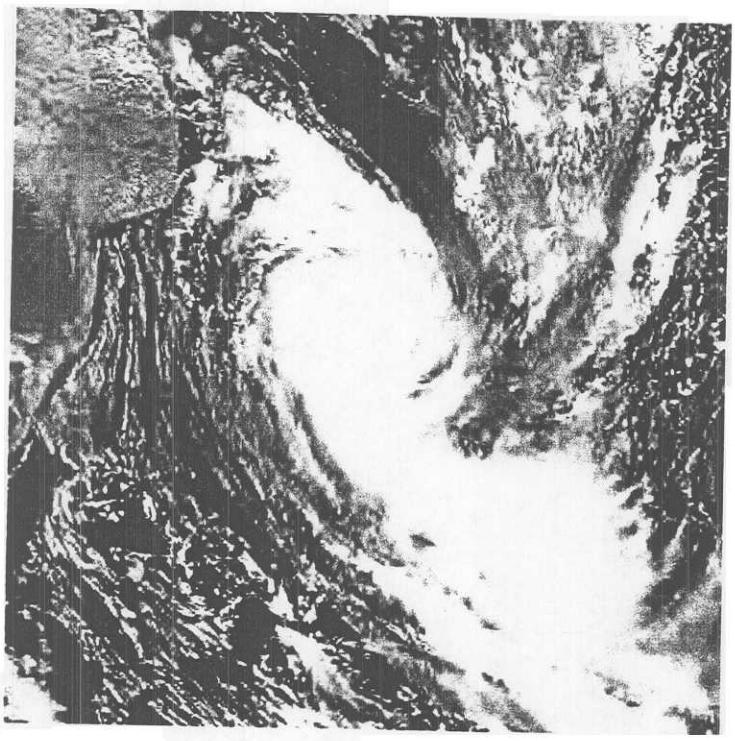
ภาพ	อัตราบิต (bpp)	PSNR (dB)	
		SPIHT	SPIHT*
Storm	0.125	22.112	22.413
	0.250	23.820	23.998
	0.500	25.918	26.174
	1.000	29.228	29.632
Florida	0.125	20.946	21.0425
	0.250	21.984	22.086
	0.500	23.869	24.080
	1.000	27.311	27.732
Mediterranean	0.125	21.962	22.173
	0.250	23.339	23.465
	0.500	25.349	25.608
	1.000	28.9724	29.330
California	0.125	21.717	21.890
	0.250	23.037	23.188
	0.500	25.068	25.396
	1.000	28.624	28.955

SPIHT* คือ SPIHT ที่ผ่านการปรับปรุง

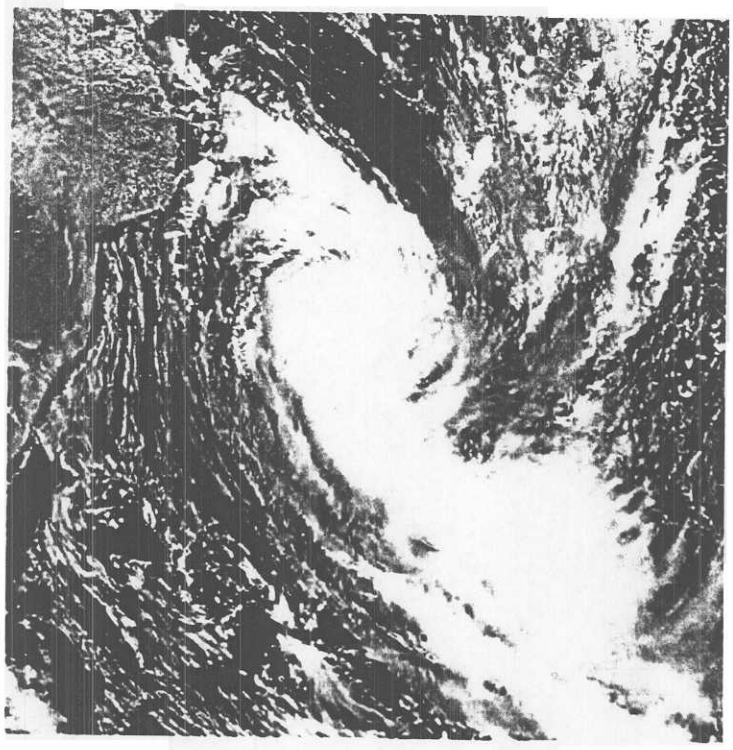
ภาพที่ 4.6.a อัตราบิต 0.125 bpp



ภาพที่ 4.6.b อัตราบิต 0.250 bpp

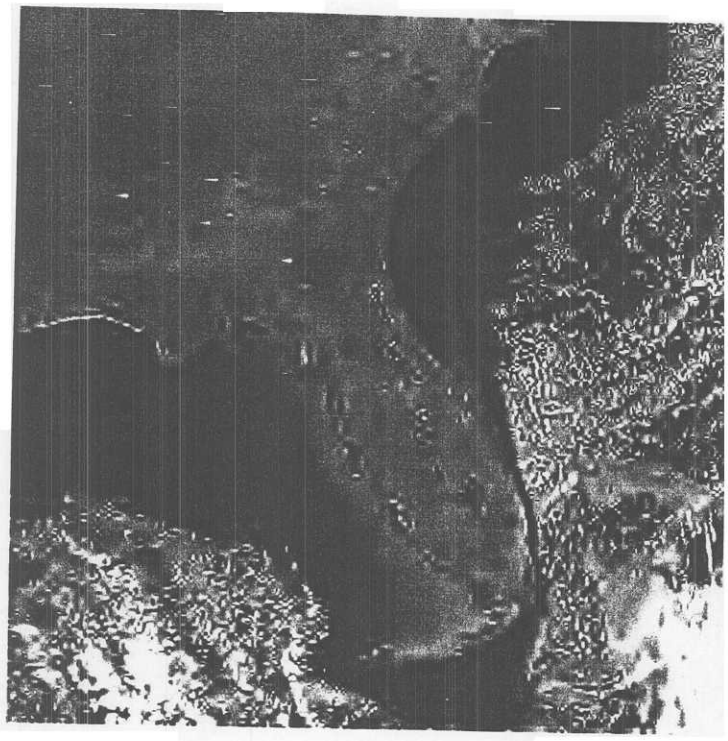


ภาพที่ 4.6.c อัตราบิต 0.500 bpp

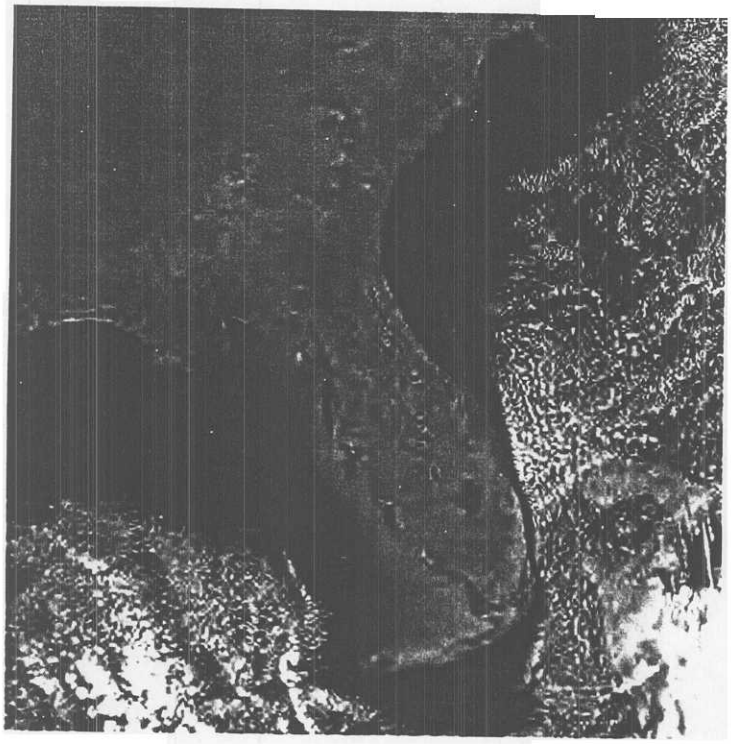


ภาพที่ 4.6.c อัตราบิต 1.000 bpp

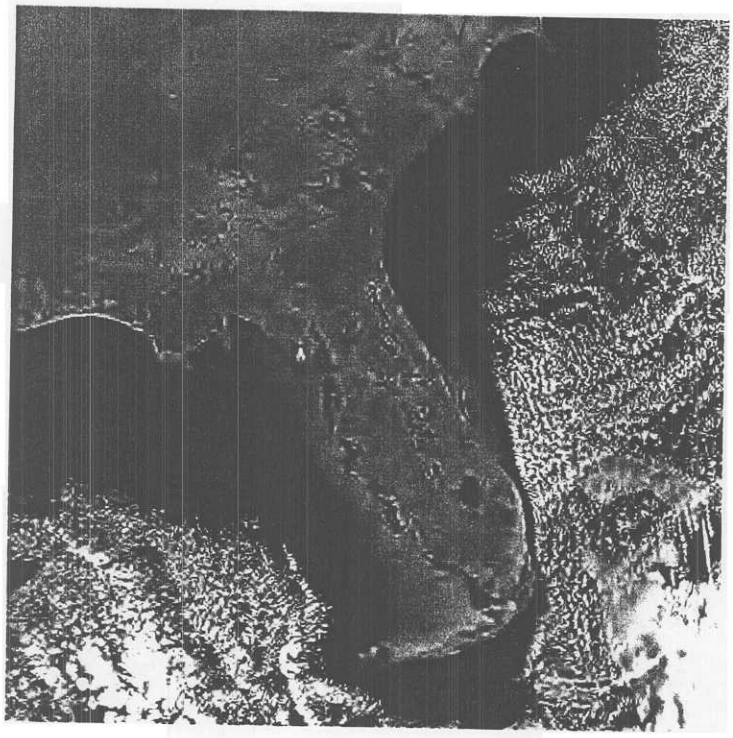
ภาที่ 4.6 ภาพ Storm ที่ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง



ภาพที่ 4.7.a อัตราบิต 0.125 bpp



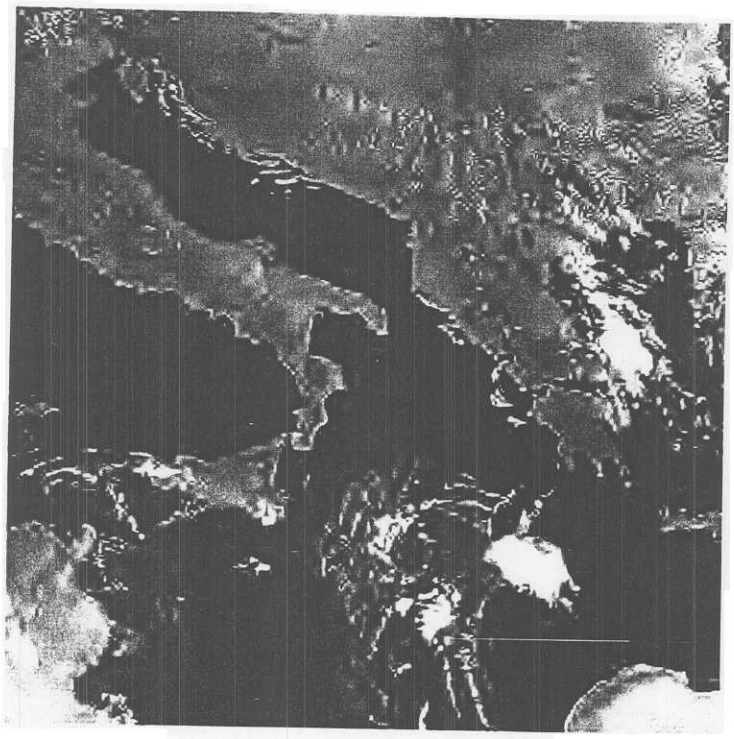
ภาพที่ 4.7.b อัตราบิต 0.250 bpp



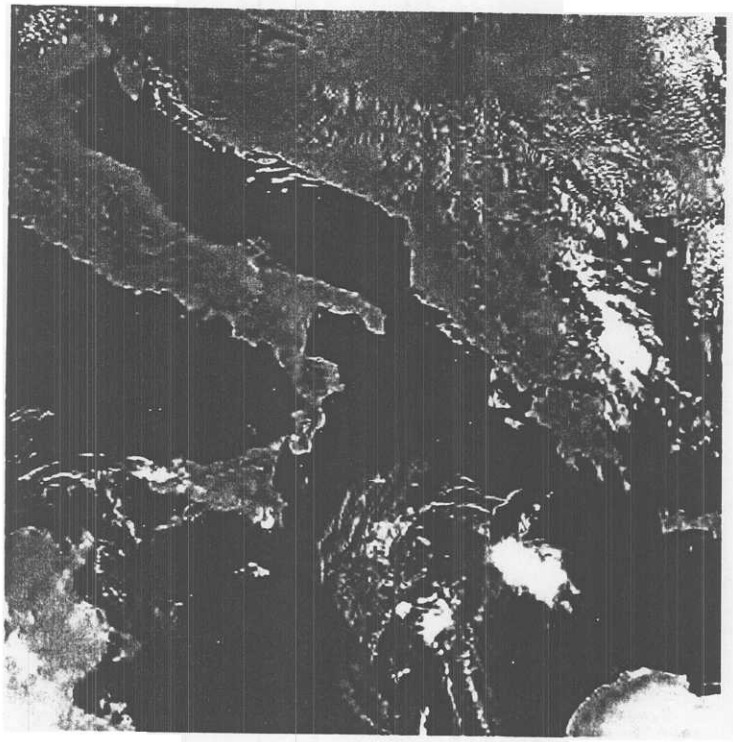
ภาพที่ 4.7.c อัตราบิต 0.500 bpp

ภาพที่ 4.7.c อัตราบิต 1.000 bpp

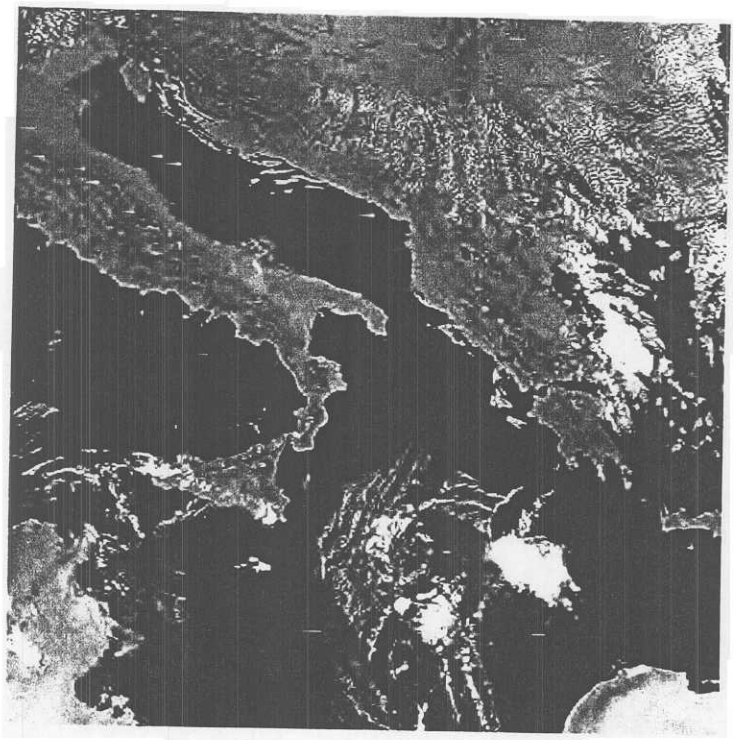
ภาที่ 4.7 ภาพ Florida ที่ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ
ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง



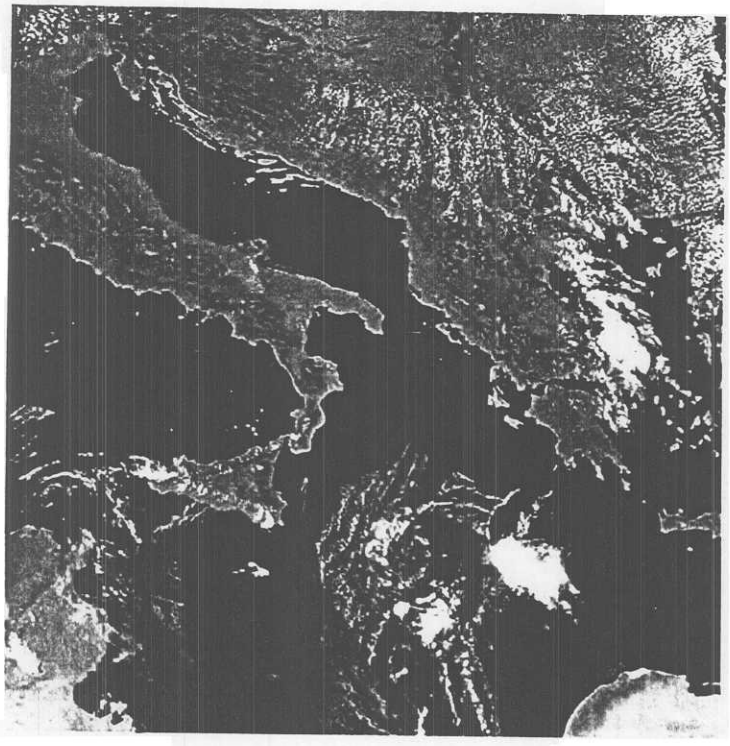
ภาพที่ 4.8.a อัตราบิต 0.125 bpp



ภาพที่ 4.8.b อัตราบิต 0.250 bpp

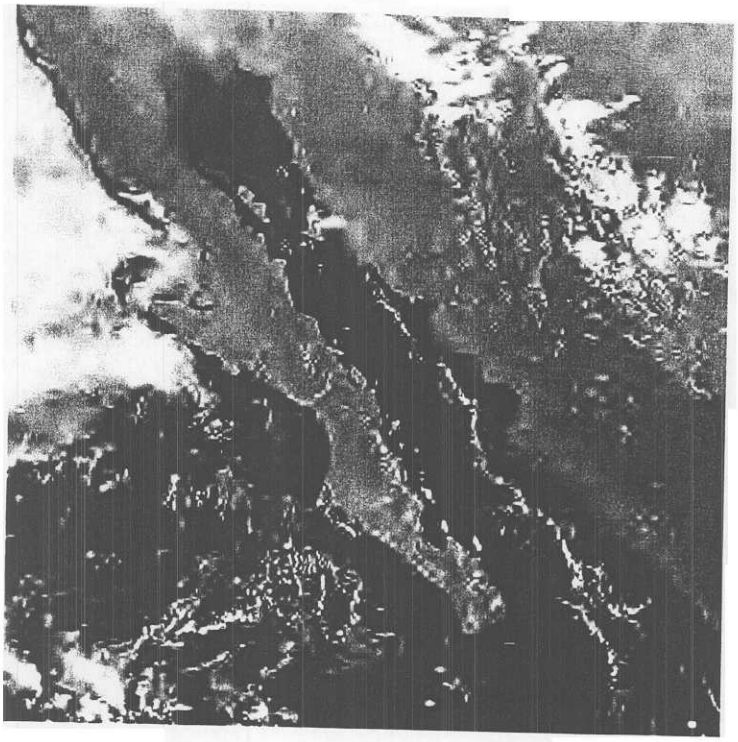


ภาพที่ 4.8.c อัตราบิต 0.500 bpp

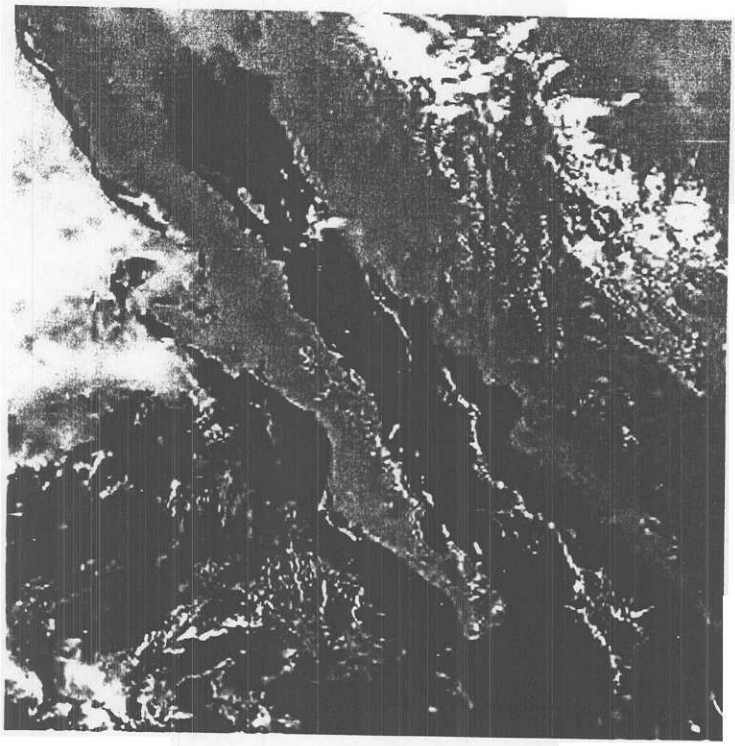


ภาพที่ 4.8.c อัตราบิต 1.000 bpp

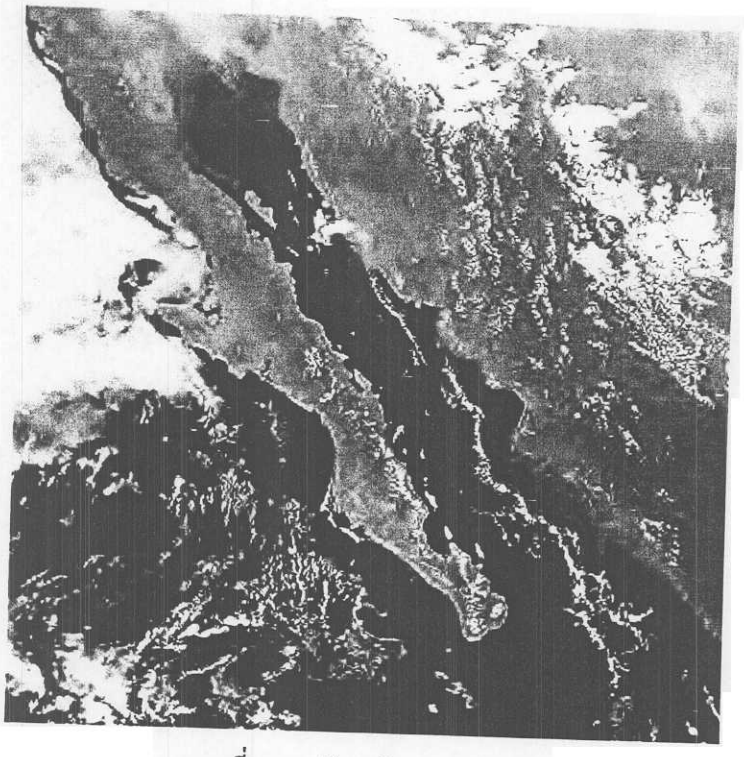
ภาที่ 4.8 ภาพ Mediterranean ที่ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง



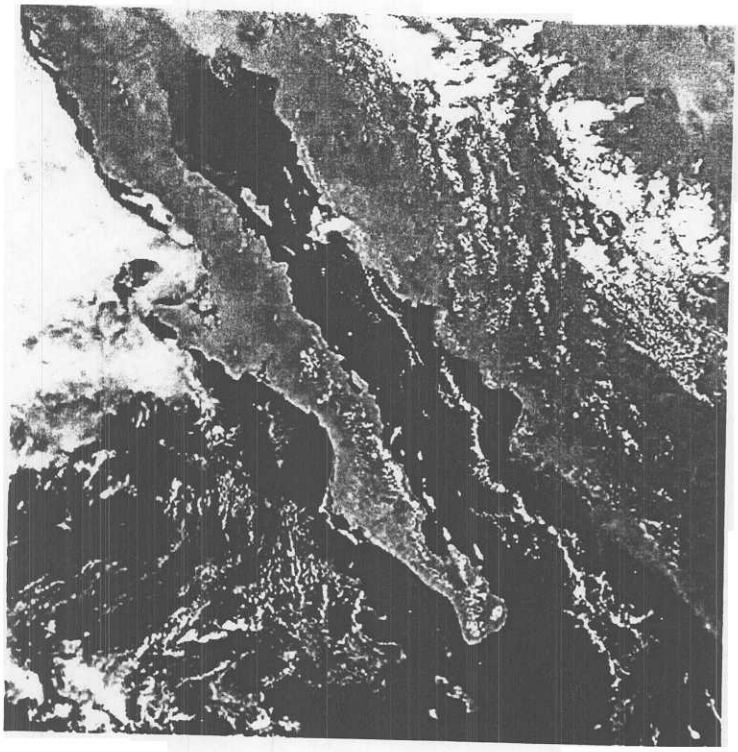
ภาพที่ 4.9.a อัตราบิต 0.125 bpp



ภาพที่ 4.9.b อัตราบิต 0.250 bpp



ภาพที่ 4.9.c อัตราบิต 0.500 bpp



ภาพที่ 4.9.c อัตราบิต 1.000 bpp

ภาที่ 4.9 ภาพ California ที่ผ่านการบีบอัดสัญญาณที่อัตราบิตต่างๆ ด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

4.7 สรุป

จากผลการทดสอบหาตระกูลเวฟเล็ตแม่ที่ให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมได้ดีที่สุด พบว่าการใช้ bi9-7 เป็นเวฟเล็ตแม่ให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมได้ดีที่สุด และจากผลการทดสอบการเปรียบเทียบประสิทธิภาพการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมด้วยอัลกอริทึม SPIHT และอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง พบว่าอัลกอริทึม SPIHT ที่ผ่านการปรับปรุงให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมได้ดีกว่า

บทที่ 5
การศึกษาเทคนิคการบีบอัดสัญญาณภาพของ JPEG200

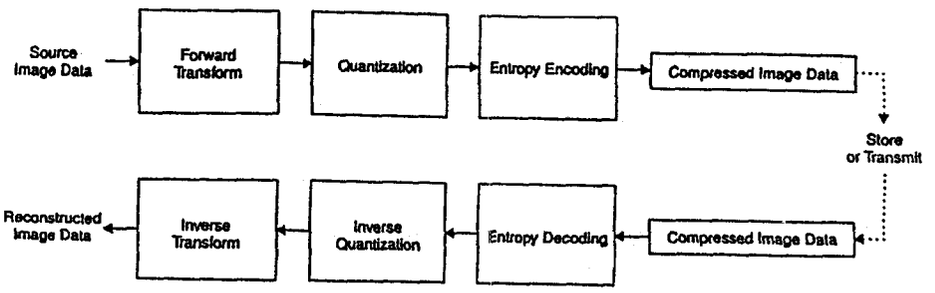
5.1 บทนำ

คณะกรรมการของ Joint Photographic Expert Group (JPEG) ได้นำมาตราฐานการเข้ารหัสภาพแบบใหม่ที่ชื่อว่า JPEG2000 มาใช้เพิ่มเติมจากมาตราฐานเดิมคือ JPEG ซึ่งมาตราฐานเดิมนี้ถูกใช้มาตั้งแต่ปี 1992 โดยมาตราฐานใหม่นี้แตกต่างจากมาตราฐานเดิมเนื่องจากมีการนำการแปลงเวฟเล็ตแบบดีสครีต (Discrete Wavelet Transform) มาใช้แทนการแปลงโคซายด์แบบดีสครีต (Discrete Cosine Transform) ซึ่งใช้ในมาตราฐานเดิมของ JPEG และวิธีการเข้ารหัสของมาตราฐานใหม่ก็แตกต่างจากมาตราฐานเดิมโดยสิ้นเชิง จึงทำให้เกิดคำถามตามมาว่า ทำไมคณะกรรมการของ JPEG ไม่พัฒนามาตราฐานเดิม แทนที่จะใช้มาตราฐานใหม่ซึ่งแตกต่างจากมาตราฐานเดิม คำตอบของคำถามนี้ก็คือ เมื่อพิจารณาช่วงเวลาที่คณะกรรม JPEG ได้เสนอการเข้ารหัสภาพครั้งแรก ในช่วงเวลานั้นทฤษฎีเวฟเล็ตยังเป็นทฤษฎีใหม่และยังไม่ได้รับการศึกษาและวิจัยอย่างลึกซึ้ง ซึ่งต่างจากทฤษฎีของการแปลงโคซายด์ซึ่งได้รับความสนใจ มีการศึกษาและวิจัยอย่างต่อเนื่องเป็นเวลานาน ดังนั้นทางคณะกรรมการจึงเลือกใช้การแปลงโคซายด์เป็นฐานในการสร้างมาตราฐานการเข้ารหัสภาพตั้งแต่ช่วงนั้นเป็นต้นมา

หลังจากที่มาตราฐานเดิมของ JPEG ถูกนำมาใช้ไม่นาน ทฤษฎีการแปลงเวฟเล็ตได้รับการวิจัยและพัฒนาไปอย่างรวดเร็วโดยเฉพาะการนำมาใช้กับการเข้ารหัสสัญญาณภาพ การคิดค้นที่สำคัญของการเข้ารหัสสัญญาณภาพเกิดขึ้นในปี ค.ศ. 1993 เมื่อ J. M. Shapiro ได้เสนอวิธีการเข้ารหัสภาพแบบใหม่โดยใช้การแปลงเวฟเล็ตแบบดีสครีต การเข้ารหัสใหม่นี้มีชื่อว่า Embedded Zero-tree Wavelet (EZW) Coding เทคนิคใหม่นี้ใช้ประโยชน์จากโครงสร้างของภาพหลังจากที่ผ่านการแปลงเวฟเล็ต ส่วนการเข้ารหัสของเทคนิคใหม่นี้มีขั้นตอนที่ไม่ซับซ้อนและที่สำคัญ ภาพที่ได้จากการถอดรหัสมีคุณภาพที่ดี ต่อมาภายหลังการเข้ารหัสสัญญาณภาพ EZW นี้เป็นพื้นฐานของการเข้ารหัสสัญญาณภาพที่ใช้การแปลงเวฟเล็ตในปัจจุบันรวมทั้งเป็นพื้นฐานของมาตราฐาน JPEG2000

5.2 มาตราฐานการเข้ารหัสภาพของ JPEG2000

มาตราฐานการเข้ารหัสสัญญาณภาพของ JPEG ได้ถูกนำมาใช้เป็นเวลาหลายปีและได้พิสูจน์ให้เห็นว่าเป็นเครื่องมือสำคัญในการลดขนาดสัญญาณภาพ ในปัจจุบันเนื่องจากการใช้สัญญาณภาพในรูปแบบของสัญญาณดิจิตอลกำลังเพิ่มขึ้นอย่างรวดเร็วและเป็นองค์ประกอบสำคัญในการใช้อินเตอร์เนต จึงมีการคิดค้นมาตราฐานใหม่ในการเข้ารหัสสัญญาณภาพให้มีประสิทธิภาพสูงขึ้นคือ



ภาพที่ 5.1 แผนภาพโครงสร้างของมาตราฐานการเข้ารหัสสัญญาณภาพ JPEG 2000

มาตราฐาน JPEG 2000 ซึ่งใช้เทคโนโลยีล่าสุดในการเข้ารหัสสัญญาณภาพและรองรับการประยุกต์ใช้สัญญาณภาพในรูปแบบของสัญญาณดิจิตอลต่างๆมากมายเช่น ภาพถ่ายทางการแพทย์ การสแกนภาพ ภาพถ่ายจากกล้องดิจิตอล อินเตอร์เนต เป็นต้น มาตราฐาน JPEG 2000 ใช้พื้นฐานของการแปลงเวฟเล็ตแบบดีสครีต การควอนไทซ์ และการเข้ารหัสเอนโทรปี ภาพที่ 5.1 แสดงแผนภาพโครงสร้างของมาตราฐานการเข้ารหัสสัญญาณภาพ JPEG 2000 ที่ตัวเข้ารหัส สัญญาณภาพจะถูกแปลงโดยใช้การแปลงเวฟเล็ต จากนั้นค่าที่ได้จะถูกนำไปผ่านขบวนการควอนไทซ์ และเข้ารหัสเอนโทรปี ส่วนที่ตัวถอดรหัสจะมีการถอดรหัสข้อมูลแบบเอนโทรปี จากนั้นจะผ่านขบวนการควอนไทซ์ย้อนกลับและการแปลงเวฟเล็ตย้อนกลับเพื่อให้ได้สัญญาณภาพกลับคืน

5.2.1 ขบวนการก่อนการเข้ารหัส (Preprocessing)

ก่อนที่จะนำภาพมาผ่านขบวนการเข้ารหัสจะมีการแบ่งสัญญาณภาพออกเป็นภาพย่อย (Image tiling) โดยภาพย่อยที่ได้จากการแบ่งจะต้องเป็นภาพสี่เหลี่ยมที่ไม่มีส่วนใดของภาพย่อยแต่ละภาพเหลื่อมกัน และภาพย่อยแต่ละภาพจะผ่านขบวนการเข้ารหัสและถอดรหัสที่เป็นอิสระต่อกัน นอกจากนี้ภาพย่อยแต่ละภาพจะต้องมีขนาดเท่ากันยกเว้นภาพย่อยที่บางตำแหน่งของภาพ เช่น บริเวณขอบภาพ ข้อดีของการแบ่งภาพเป็นภาพย่อยคือ การลดขนาดของหน่วยความจำที่ต้องใช้ ถ้าภาพย่อยมีขนาดใหญ่ เราจำเป็นต้องใช้หน่วยความจำมาก อย่างไรก็ตามเราพบว่าการแบ่งภาพออกเป็นภาพย่อยจะลดคุณภาพของภาพที่ได้จากการถอดรหัส ยิ่งแบ่งภาพให้เป็นภาพย่อยขนาดเล็กลงเท่าไร คุณภาพของภาพที่ได้จากการถอดรหัสก็ยิ่งลดลง ในการเลือกขนาดภาพย่อย เราสามารถกำหนดให้ภาพย่อยมีขนาดเท่ากับภาพที่เราต้องการเข้ารหัสหรือภาพเริ่มต้น ซึ่งก็คือจำนวนภาพย่อยเท่ากับหนึ่ง หรือเราสามารถนำภาพเริ่มต้นเข้าขบวนการเข้ารหัสได้เลยโดยไม่ต้องแบ่งเป็นภาพย่อย กรณีนี้เป็นกรณีที่เราต้องการคุณภาพของภาพสูงและมีหน่วยความจำมากพอ หลังจากที่แบ่งภาพออกเป็นภาพย่อย ภาพย่อยแต่ละภาพจะผ่านขบวนการปรับระดับ ดีซี (DC level shifting) คือการลดค่าของจุด

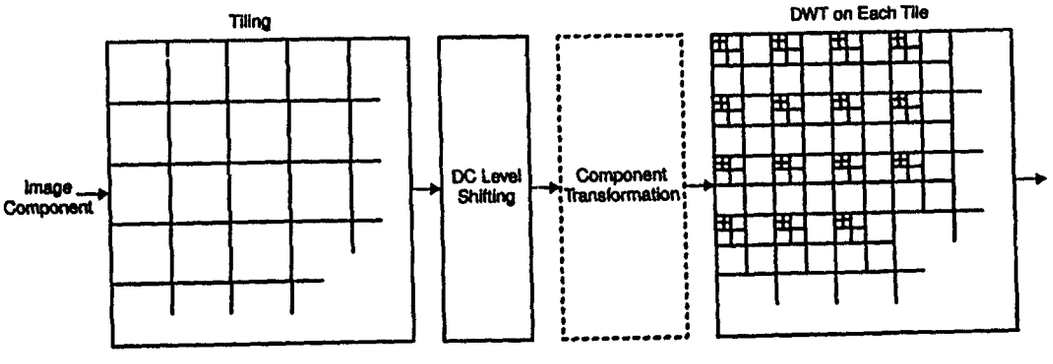
ภาพแต่ละจุดภาพในภาพย่อยด้วยค่าที่เท่าๆกัน กระบวนการปรับระดับดีซีของภาพจะกระทำกับภาพที่ค่าของจุดภาพมีเฉพาะค่าบวก เพื่อให้ค่าของจุดภาพหลังจากการปรับระดับดีซีมีทั้งค่าบวกและลบก่อนที่จะเข้าขบวนการเข้ารหัส ถ้ามีการปรับระดับดีซีก่อนการเข้ารหัส ต้องมีการคืนระดับดีซีหลังจากการถอดรหัส

ในขบวนการก่อนการเข้ารหัส นอกเหนือจากการแบ่งภาพเป็นภาพย่อย การปรับระดับดีซีแล้ว ยังมีกระบวนการแปลงองค์ประกอบของภาพซึ่งเป็นขบวนการที่ไม่บังคับจะทำหรือไม่ทำก็ได้ การแปลงองค์ประกอปของภาพมี 2 วิธี คือ กระบวนการแปลงองค์ประกอบของภาพแบบย้อนกลับไม่ได้ (Irreversible component transformation) ซึ่งใช้กับการเข้ารหัสภาพที่มีการสูญเสีย และกระบวนการแปลงองค์ประกอบของภาพแบบย้อนกลับได้ (Reversible component transformation) ซึ่งใช้กับการเข้ารหัสภาพที่ไม่มีการสูญเสีย ภาพที่ 5.2 แสดงแผนภาพของขบวนการก่อนการเข้ารหัส

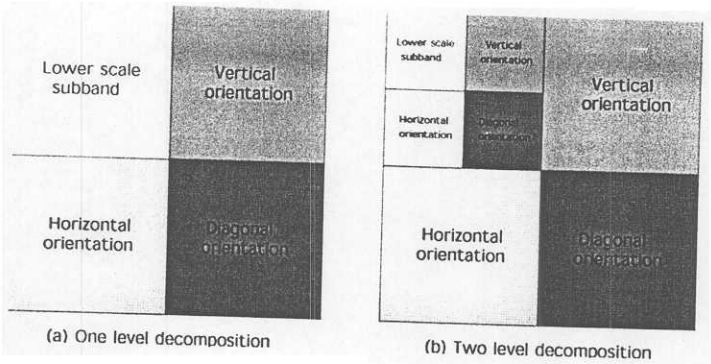
5.2.2 การเข้ารหัสและถอดรหัสของมาตราฐาน JPEG 2000

1. การแปลงเวฟเล็ตแบบดีสครีต

การแปลงเวฟเล็ตจะถูกนำมาใช้กับภาพย่อยแต่ละภาพ โดยการใช้การแปลงเวฟเล็ตหนึ่งครั้งเรียกว่า การแยกย่อยหนึ่งระดับ ซึ่งการแยกย่อยระดับแรกจะให้แบนด์ย่อยของภาพจำนวนสี่แบนด์ย่อยและแต่ละแบนด์ย่อยจะให้สัมประสิทธิ์เวฟเล็ตที่เกี่ยวข้องกับข้อมูลภาพทางด้านแนวนอน แนวตั้ง แนวทแยง และข้อมูลภาพความถี่ต่ำ ในการแยกย่อยของการแปลงเวฟเล็ตระดับต่อๆไป เราจะใช้ข้อมูลภาพความถี่ต่ำของระดับการแยกย่อยก่อนหน้านี้มาเป็นอินพุทของการแปลงเวฟเล็ต ภาพที่ 5.3 แสดงการแบ่งแบนด์ย่อยของภาพจากการแปลงเวฟเล็ตโดยใช้การแยกย่อยหนึ่งระดับและสองระดับ ภาพที่ 5.4 แสดงภาพ Lena และผลจากการแปลงเวฟเล็ตโดยใช้การแยกย่อยสองระดับ



ภาพที่ 5.2 แผนภาพของขบวนการก่อนการเข้ารหัส



ภาพที่ 5.3 แผนภาพการแบ่งแบนด์ย่อยของภาพจากการแปลงเวฟเล็ตโดยใช้การแยกย่อยหนึ่งระดับและสองระดับ

ภาพที่ 5.4 ภาพ Lena และผลจากการแปลงเวฟเล็ตโดยใช้การแยกย่อยสองระดับ

2. การควอนไทซ์

หลังจากที่ภาพย่อยถูกแปลงด้วยการแปลงเวฟเล็ต สัมประสิทธิ์ของเวฟเล็ตที่ได้จะผ่านขบวนการควอนไทซ์แบบสเกลาร์สม่ำเสมอ (Uniform scalar quantization) โดยสัมประสิทธิ์เวฟเล็ต $a_b(u,v)$ จากแบนด์ย่อย b จะถูกควอนไทซ์ให้มีค่าเป็น $q_b(u,v)$ จากสูตร

$$q_b(u,v) = sign(a_b(u,v)\left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor$$

โดยที่ Δ_b คือความกว้างของช่วงการควอนไทซ์ ของแบนด์ย่อย b ดังนั้นเราจะเห็นได้ว่ามาตราฐาน JPEG2000 สามารถกำหนดความกว้างของการควอนไทซ์สำหรับแบนด์ย่อยแต่ละแบนด์ อย่างไรก็ตาม สัมประสิทธิ์เวฟเล็ต ภายในแบนด์ย่อยเดียวกันจะต้องใช้ความกว้างของการควอนไทซ์ที่เท่ากัน โดยความกว้างของการควอนไทซ์จะขึ้นอยู่กับจำนวนบิทที่ใช้แทนจุดภาพในภาพย่อยเริ่มต้นและชนิดของเวฟเล็ตที่ใช้ หลังจากที่สัมประสิทธิ์เวฟเล็ตผ่านกระบวนการควอนไทซ์ จะมีการเรียงลำดับของสัมประสิทธิ์ที่มีความสำคัญมากโดยจะทำการเข้ารหัสกับสัมประสิทธิ์ที่มีความสำคัญมากก่อน ความสำคัญของสัมประสิทธิ์ขึ้นกับขนาดของสัมประสิทธิ์ สัมประสิทธิ์ที่มีขนาดใหญ่จะมีความสำคัญมากกว่าสัมประสิทธิ์ที่มีขนาดเล็กกว่า การเรียงลำดับความสำคัญของสัมประสิทธิ์อาศัยโครงสร้างของแบนด์ย่อยที่ได้หลังจากการใช้การแปลงเวฟเล็ต ซึ่งการเรียงลำดับความสำคัญนี้จะใช้เทคนิคพิเศษที่ไม่ต้องนำข้อมูลของตำแหน่งสัมประสิทธิ์แต่ละตัวมาเข้ารหัส และใช้สมมุติฐานที่ว่าถ้าสัมประสิทธิ์เวฟเล็ตในระดับการแยกย่อยสูงมีค่าน้อย มีความเป็นไปได้สูงที่ค่าสัมประสิทธิ์เวฟเล็ตที่ตำแหน่งเดียวกันในระดับการแยกย่อยที่ต่ำกว่าก็มีค่าน้อยเช่นกัน ซึ่งก็คือพื้นฐานของการเข้ารหัสสัญญาณภาพแบบ EZW

3. การเข้ารหัสเอนโทรปี

สำหรับมาตราฐาน JPEG2000 การเข้ารหัสเอนโทรปีจะใช้เทคนิคการเข้ารหัสเรขาคณิต (Arithmetic coding) ในการบีบอัดสัญญาณไบนารีซึ่งใช้ความน่าจะเป็นแบบปรับตัวได้ของการเกิดขึ้นของสัญญาณไบนารีแต่ละตัว

ส่วนขั้นตอนการถอดรหัสจะใช้ขบวนการย้อนกลับของแต่ละขั้นตอน โดยเริ่มจากการถอดรหัสของการเข้ารหัสเรขาคณิต การทำการดีควอนไทซ์ และการแปลงเวฟเล็ตย้อนกลับ (Inverse wavelet transform) เพื่อให้ได้ภาพย่อยกลับคืนมา ถ้าเราเลือกการเข้ารหัสสัญญาณภาพแบบไม่สูญเสีย (Lossless image coding) ภาพย่อยที่ได้กลับมาและภาพย่อยเริ่มต้นจะมีจุดภาพที่ตำแหน่งเดียวกันมีค่าเท่ากัน สำหรับการเข้ารหัสสัญญาณภาพแบบสูญเสีย (Lossy image coding) ภาพย่อยที่ได้กลับมาและภาพย่อยเริ่มต้นจะมีจุดภาพที่ตำแหน่งเดียวกันมีค่าไม่เท่ากัน

5.3 ข้อดีของการเข้ารหัสสัญญาณภาพโดยใช้มาตราฐาน JPEG2000

1. มาตราฐาน JPEG2000 ให้คุณภาพของภาพดีกว่ามาตราฐานเดิมของ JPEG โดยเฉพาะเมื่อเปรียบเทียบที่บิทเรตค่าต่ำ ตัวอย่างเช่นที่ บิทเรตต่ำกว่า 0.25 bpp สำหรับสัญญาณภาพระดับเทา (Gray-scaled image)

2. มาตราฐาน JPEG2000 สามารถเลือก การเข้ารหัสเฉพาะบางส่วนของภาพ (Region-of-interest coding) ในการเข้ารหัสภาพ บางครั้งบางส่วนของภาพมีความสำคัญมากกว่าส่วนอื่นๆ ดังนั้นเราสามารถเลือกการเข้ารหัสส่วนที่สำคัญนั้นให้คุณภาพของภาพดีกว่าส่วนอื่นๆที่สำคัญน้อยกว่า

3. การส่งข้อมูลสัญญาณภาพที่ถูกบีบอัดเป็นแบบก้าวหน้า (Progressive image transmission) คือ ไม่ว่าตัวถอดรหัสจะรับข้อมูลมาเท่าไดก็สามารถแปลงเป็นสัญญาณภาพขนาดเท่ากับสัญญาณภาพเริ่มต้นได้ โดยคุณภาพของภาพจะดีขึ้นตามจำนวนข้อมูลที่ได้รับจากตัวเข้ารหัส เช่น ถ้ารับข้อมูลจากตัวเข้ารหัสน้อย คุณภาพของภาพที่ได้จากตัวถอดรหัสจะต่ำ แต่เวลาที่ใช้ในการถอดรหัสจะเร็ว ถ้ารับข้อมูลจากตัวเข้ารหัสจำนวนมาก คุณภาพของภาพจะดีเพิ่มขึ้น แต่เวลาที่ใช้ในการถอดรหัสจะช้ากว่า คุณสมบัตินี้มีประโยชน์ในการเลือกดูภาพ เราสามารถเลือกดูภาพโดยการรับข้อมูลจำนวนน้อยๆ จากตัวเข้ารหัส เพื่อตรวจสอบว่าเป็นภาพที่ต้องการหรือไม่ ถ้าเป็นภาพที่ต้องการ เราสามารถสั่งตัวถอดรหัสให้รับข้อมูลเพิ่มขึ้น ซึ่งทำให้ประหยัดเวลาในการเลือกสัญญาณภาพ

5.4 สรุป

จากการศึกษามาตราฐานการเข้ารหัสสัญญาณภาพ JPEG2000 ในเบื้องต้น เราพบว่า หัวใจสำคัญของการเข้ารหัสสัญญาณภาพแบบนี้คือ การแปลงสัญญาณภาพโดยใช้การแปลงเวฟเล็ต การเข้ารหัสโดยใช้ประโยชน์จากโครงสร้างของแบนด์ย่อยที่ได้จากการแปลงเวฟเล็ต และสมมุติฐานที่ว่าถ้าสัมประสิทธิ์เวฟเล็ตในระดับการแยกย่อยสูงมีค่าน้อย มีความเป็นไปได้สูงที่ค่าสัมประสิทธิ์เวฟเล็ตที่ตำแหน่งเดียวกันในระดับการแยกย่อยที่ต่ำกว่าก็มีค่าน้อยเช่นกัน และจากการศึกษาข้อดีของมาตราฐานใหม่ เราพบว่า มาตราฐานใหม่มีข้อดีกว่ามาตราฐานเดิมทั้งทางด้านคุณภาพของภาพและคุณสมบัติอื่นๆที่ไม่มีในมาตราฐานเดิมเช่น การส่งข้อมูลสัญญาณภาพที่ถูกบีบอัดเป็นแบบก้าวหน้า

บทที่ 6
สรุปและข้อเสนอแนะ

6.1 บทนำ

งานวิจัยนี้ได้ทำการศึกษาการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม เพื่อลดขนาดของสัญญาณ ในการส่งข้อมูลจากดาวเทียมไปยังสถานีภาคพื้นดินให้ได้มากขึ้นในเวลาเท่าเดิม งานวิจัยนี้นำเสนอวิธีการบีบอัดสัญญาณภาพที่เหมาะสำหรับภาพถ่ายจากดาวเทียม ด้วยวิธีการแปลงเวฟเล็ตแบบดีสครีท โดยภาพถ่ายจากดาวเทียมเมื่อนำมาแปลงด้วยวิธีเวฟเล็ตแบบดีสครีท สัญญาณที่เป็นพื้นผิววัตถุจะอยู่ที่สัมประสิทธิ์ที่แบนด์ย่อยความถี่ต่ำ และสัญญาณที่เป็นขอบหรือลายเส้นของวัตถุจะอยู่ที่สัมประสิทธิ์ที่แบนด์ย่อยความถี่สูง ดังนั้นการบีบอัดสัญญาณภาพถ่ายจากดาวเทียม ด้วยการแปลงเวฟเล็ตจะต้องให้ความสำคัญกับสัมประสิทธิ์ทุกแบนด์ย่อย เพื่อให้ภาพที่ได้จากการบีบอัดมีความใกล้เคียงกับภาพต้นแบบมากที่สุด อัลกอริทึม SPIHT เป็นอัลกอริทึมหนึ่งที่เหมาะสม เนื่องจากอัลกอริทึมนี้เข้ารหัสโดยให้ความสำคัญกับขนาดของสัมประสิทธิ์และไม่สนใจว่าสัมประสิทธิ์ตัวนั้นจะอยู่ในระดับแบนด์ย่อยใด ในงานวิจัยนี้ยังเสนอวิธีการพัฒนาอัลกอริทึม SPIHT โดยการเพิ่ม LFC และเพิ่มเงื่อนไขการเข้ารหัสและการถอดรหัส และทำการหาตระกูลเวฟเล็ตแม่ที่ให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมได้ดีที่สุด นอกจากนี้งานวิจัยนี้ยังศึกษาเทคนิคการเข้ารหัสสัญญาณภาพของมาตราฐาน JPEG2000 เบื้องต้น เพื่อเป็นพื้นฐานในการศึกษาและวิจัยเกี่ยวกับมาตราฐานการเข้ารหัสสัญญาณภาพต่อไป

6.2 สรุป

การดำเนินงานวิจัยนี้บรรลุวัตถุประสงค์ ตามที่ตั้งไว้ทุกประการ มีผลการวิจัยคือ การบีบอัดสัญญาณภาพถ่ายจากดาวเทียม ด้วยการแปลงเวฟเล็ตแบบดีสครีท ที่ใช้ bi9-7 เป็นเวฟเล็ตแม่ในการแปลงเวฟเล็ต แล้วนำสัมประสิทธิ์เวฟเล็ตมาเข้ารหัสด้วยอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง จะให้ผลการบีบอัดสัญญาณภาพถ่ายจากดาวเทียมได้ดีที่สุด

6.3 ข้อเสนอแนะ

1. ในการประยุกต์ใช้กับภาพชนิดอื่น ภาพที่จะนำมาประยุกต์ใช้ต้องเป็นภาพระดับเทา ในกรณีต้องการบีบอัดภาพสีจะต้องทำการศึกษาเพิ่มเติม เพราะโครงสร้างของไฟล์ภาพสีกับภาพระดับเทามีโครงสร้างไม่เหมือนกัน

2. โปรแกรมการบีบอัดสัญญาณด้วยอัลกอริทึม SPIHT ที่ทำการปรับปรุงเป็นเพียงโปรแกรมที่เขียนเพื่อทดสอบอัลกอริทึม SPIHT ที่ทำการปรับปรุงเท่านั้น ทำให้โปรแกรมที่ได้จะใช้เวลาในการ

บีบอัดสัญญาณค่อนข้างนานเมื่อเทียบกับการบีบอัดสัญญาณด้วยวิธี JPEG ดังนั้นต้องทำการพัฒนาการเขียนโปรแกรมการบีบอัดสัญญาณ เพื่อให้ใช้เวลาในการบีบอัดสัญญาณน้อยลง

บรรณานุกรม

ขวัญฤทัย ไพรีพ่ายฤทธิ์. (2542). ตระกูลเวฟเลตที่เหมาะสมสำหรับการเข้ารหัสแบบเอ็มเบตเด็ตซีโร่ทรีเวฟเล็ตในการประยุกย์ใช้งานการแพทย์ทางไกล. วิทยานิพนธ์ปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าธนบุรี.

ศิริพร เดชะศิลารักษ์. (2543). การลดขอบบล็อคในภาพ JPEG ด้วยวิธีเวฟเล็ตเธรโชลดิ้งที่ปรับตัวเองได้. วิทยานิพนธ์ปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าคุณทหารลาดกระบัง.

สุขสันต์ จิรเชวง และ วุฒิพงศ์ อารกุล (2544). การปรับปรุงการบีบอัดภาพวิธี SPIHT โดยใช้การเข้ารหัสบล็อกสำหรับภาพลายเส้นคม. การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 24 (EECON-24) สจล. 1214-1219.

Al, B. (2000). Handbook of image & video processing. New York: Academic Press.

Amir, S., and William, A. P. (1993). Image compression using the spatial-orientation tree. IEEE Int. Symp. On Circuits and Systems, Chicaro, IL, 279-282.

Amir, S., and William, A. P. (1996). A new fast and efficient image codec based on set partitioning in hierarchical trees. IEEE Transaction on Circuit and Systems for Video Technology. 6 (3): 243-250.

Brian, A. B., and Thomas, R. F. (2001). Quadtree classification and TCQ image coding. IEEE Transactions on Circuit and Systems for Video Technology. 11 (1): 3-8.

Jabran, A. (2001). Optimization of biorthogonal wavelet filters for signal and image compression. Degree of Candiatus Scientarium at the Department of Informatics: University of Oslo.

Jerome, M. S. (1993). Embedded image coding using zerotrees of wavelets coefficients. IEEE Transaction on Signal Processing. 41 (12): 3445-3462.

Marc, A., Michel, B., Pierre, M., and Ingrid, D. (1992). Image coding using wavelet transform. IEEE Transactions on Image Processing. 1 (2): 205-220.

Michel, M., Yves, M., Georges, O., and Jean, M. P. (1996). Wavelet toolbox for use with matlab. New York: The MathWorks.

Olivier, R., and Martin, V. (1991). Wavelets and signal processing. IEEE SP Magazine. 14-38.

Stephane, G. M. (1989). A theory for multiresolution signal decomposition: The wavelet representation. IEEE Transaction on Pattern and Machine Intelligence. 11 (7): 674-693.

ภาคผนวก ก

รายละเอียดโปรแกรมการเข้ารหัสอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

โปรแกรม SPIHT.h

```
/*-------------------------------------------------------------*/
//DWT and IDWT (1D and 2D) -> bi9-7
void conv(float* out,float* hn,float* xn,int size_h,int
size_x);
void conv_p(float* hn,float* xn,int size_x,int size_h);
//DWT
void refind_data_bi9(float* out,float* in,int size_x);
void refind_data_bi7(float* out,float* in,int size_x);
void bi_dwt(float* inx,int size_x);
void down_sampling(float* in,int st,int nxt,int type);
void bi_dwt2(float** fdwt,int s_x,int s_y);
//IDWT
void bi_idwt(float* inx,int size_x);
void up_sampling(float* out,float* in,int size_x,int type);
void bi_idwt2(float** fdwt,int s_x,int s_y);
//Encode SPIHT
void coor_offspring(int** O_list,int i,int j,int s_x,int s_y,int
level);
void coor_descendants(int*,int i,int j,float threshold,float**
fdwt,int s_x,int s_y,int level);
//LFC SPIHT
void LFC_list(int* q_LFC,float** image_in,int s_x,int
s_y,int level);
void cut_image(float** output,float** input,int x_start,int
x_end,int y_start,int y_end);
void dis_max_abs(float* max_out,float** input,int ss_x,int
ss_y);
int type_label(int x,int y,int s_x,int s_y,int level);
int floor_LFC_n(float max_out_n,int max_n);
int log2_floor(float data);
void sn_out(float* output,int* LIP_x,int* LIP_y,int t,float**
fdwt,float num_n);
void cut_LIP(long* count_LIP_new,int* LIP_x,int*
LIP_y,long count_LIP);
void cut_LIS(long* count_LIS_new,int* LIS_x,int*
LIS_y,int* LIS_type,long count_LIS);
void refinement_pass(long* count_new,char* imout,int*
LSP_abs,long count_LSP,float num_n,long bit_end,long
count);
int find_max_coff(float** fdwt,int s_x,int s_y,int level);
//SPIHT
void SPIHT_Encode(char* LFC_list_type,long*
countp,char* imout,int end_bit,float** fdwt,int s_x,int
s_y,int level);
void SPIHT_Decode(char* LFC_list_type,float**
output_image,char* imout,int bit_end,int s_x,int s_y,int
level,int n);
void i_coor_descendants(int** O_list,int* type_ij,int i,int
j,int s_x,int s_y,int level);
void i_refinement_pass(long* count_new,char* imout,int*
LSP_abs,long count_LSP,float num_n,long bit_end,long
count);
void re_image(int** output_image,int* LSP_x,int*
LSP_y,int* LSP_abs,int* LSP_sign,long count_LSP);
void coor_descendants_s(int* type_ij,int i,int j,float
threshold,float** fdwt,int s_x,int s_y,int level);
void coor_offspring_s(int* countO,int** O_list,int i,int j,int
s_x,int s_y,int level);
void i_coor_descendants_s(int *count_set,int** O_list,int*
type_ij,int i,int j,int s_x,int s_y,int level);
void round_image_pass(float** input,int s_x,int s_y,float
bitrate);
void round_image_abs(float** input,int s_x,int s_y);
void re_image_pass(float** output_image,int* LSP_x,int*
LSP_y,int * LSP_abs,long count_LSP);
```

โปรแกรม coff.h

```
/*-------------------------------------------------------------*/
/*decomposition lowpass*/
#define bdl0 0.03782879857992
#define bdl1 -0.02384929751586
#define bdl2 -0.11062402748951
#define bdl3 0.37740268810913
#define bdl4 0.85269865321930
#define bdl5 0.37740268810913
#define bdl6 -0.11062402748951
#define bdl7 -0.02384929751586
#define bdl8 0.03782879857992
/*decomposition highpass*/
#define bdh0 0.06453905013246
#define bdh1 -0.04068975261660
#define bdh2 -0.41809244072573
#define bdh3 0.78848487220618
```

```
#define bdh4 -0.41809244072573
#define bdh5 -0.04068975261660
#define bdh6 0.06453905013246
/*reconsturction lowpass*/
#define brl0 -0.06453905013246
#define brl1 -0.04068975261660
#define brl2 0.41809244072573
#define brl3 0.78848487220618
#define brl4 0.41809244072573
#define brl5 -0.04068975261660
#define brl6 -0.06453905013246
/*reconsturction highpass*/
#define brh0 0.03782879857992
#define brh1 0.02384929751586
#define brh2 -0.11062402748951
#define brh3 -0.37740268810913
#define brh4 0.85269865321930
#define brh5 -0.37740268810913
#define brh6 -0.11062402748951
#define brh7 0.02384929751586
#define brh8 0.03782879857992
```

โปรแกรม constant.h

```
/*-------------------------------------------------------------*/
////////// 2-D ////////////////////////////////////////
//input and DWT,IDWT
#define r_xx 960   //(row-Image-960)
#define r_yy 1280  //(column-Image-1280)
////////////////////////////////////////////////////////
#define level_s 5        //Level of Wavelet Tranfrom
#define max_nr -1        //Check No root of o(i,j)
#define step_quan 12  //Set step_Quantization
```

โปรแกรม main.cpp

```
/*-------------------------------------------------------------*/
#include"SPIHT.h"
#include"coff.h"
main(){
////////////////////////////////////////////////////////////////
//* Transfer Data from VC++      to MATLAB
// Set r_x,r_y (size Data input)
////////////////////////////////////////////////////////////////
FILE *p_read;  long int numread;
long int count_r = 0; int i,r_i,r_j;
//generate size pointer <Dynamic >
unsigned char *point_read = new unsigned char[r_xx*r_yy];
//for DATA IMAGE
float **fdwt = new float*[r_xx];
float **imin = new float*[r_xx];
for(  i=0; i<r_xx; i++)
{fdwt[i] = new float[r_yy]; imin[i] = new float[r_yy];}
p_read = fopen("histioc1.dat","rb");
numread = fread(point_read, sizeof(char), r_xx*r_yy ,
p_read);
for (r_i = 0; r_i < r_yy ; r_i++)
{for (r_j = 0; r_j < r_xx ; r_j++)
{fdwt[r_j][r_i] = (float) point_read[count_r];
count_r = count_r + 1;}}
count_r = 0;
for (r_i = 0; r_i < r_yy ; r_i++)
{for (r_j = 0; r_j < r_xx ; r_j++)
{min[r_j][r_i] = (float)point_read[count_r];
count_r = count_r + 1;}}
fclose(p_read);  delete[] point_read;
//// DWT
bi_dwt2(fdwt,r_xx,r_yy);
bi_dwt2(fdwt,r_xx/2,r_yy/2);
bi_dwt2(fdwt,r_xx/4,r_yy/4);
bi_dwt2(fdwt,r_xx/8,r_yy/8);
bi_dwt2(fdwt,r_xx/16,r_yy/16);
////////////////////////////////////////////////////////////////
* SPIHT -> Encode
////////////////////////////////////////////////////////////////*/
//DATA HEADER
int s_x = r_xx; //->Dimension X
int s_y = r_yy; //->Dimension Y
int level = level_s; //-> level wavelet tranfrom
//Initial DATA SPIHT = end_bit (bit)
int bit_end,n; float bitrate;
printf("bit rate = ");
scanf("%f",&bitrate);
bit_end = (int)(s_x*s_y*bitrate);
char* imout = new char[bit_end];
```

```
n = find_max_coff(fdwt,s_x,s_y,level);
long* countp = new long[1];
//LFC List
int count_LFC;
count_LFC = (level*4*3) + 4;
char* LFC_list_type = new char[count_LFC];
//Encode SPIHT + LFC
round_image_pass(fdwt,r_xx,r_yy,bitrate);
SPIHT_Encode(LFC_list_type,countp,imout,bit_end,fdwt,s_
x,s_y,level);
////////////////////////////////////////////////////////////////////
// save data
FILE *pimout_write;
// Open file in SUT mode:
pimout_write = fopen("output.sut","wb");
 //generate size pointer <Dynamic >
fwrite(imout,sizeof(char),bit_end,pimout_write);
fclose( pimout_write );
FILE *pheader; pheader = fopen("header.sim","wb");
int header[1];
header[0] = countp[0];   //size of data wavelet SPIHT
fwrite(header,sizeof(int),1,pheader);
header[0] = s_x ;   //hight image
fwrite(header,sizeof(int),1,pheader);
header[0] = s_y ;   //wide image
fwrite(header,sizeof(int),1,pheader);
header[0] = level ; //level decomposition
fwrite(header,sizeof(int),1,pheader);
header[0] = n;      //number step Quantization
fwrite(header,sizeof(int),1,pheader);
fwrite(LFC_list_type,1,count_LFC,pheader);
fclose(pheader);
return(0);}
```

โปรแกรม bi_dwt.cpp

```
/*-------------------------------------------------------------*/
#include"SPIHT.h"
#include"coff.h"
////////////////////////////////////////////////
//Bi-orthrogonal 9-7
////////////////////////////////////////////////
void bi_idwt2(float** fdwt,int s_x,int s_y){
% Reconstruction 2-D Wavelet Tranform
% yA is input lowpass
% yD is input highpass
% CL is filter cofficient (lowpass)
% CH is filter cofficient (highpass)
% y is Image reconstruction output*/
long int b,g; float *ll = new float[s_x];
float *hh = new float[s_y];
//% columns image
for (b=0; b<s_y ; b++)
{ for(g=0; g<s_x ; g++){ll[g] = fdwt[g][b];}
bi_idwt(ll,s_x);
for(g=0; g<s_x  ; g++){fdwt[g][b] = ll[g];}}
//% rows image
for (b=0 ; b<s_x ; b++)
{for(g=0 ; g<s_y ; g++){hh[g] = fdwt[b][g];}
bi_idwt(hh,s_y);
for(g=0 ; g<s_y ;g++){fdwt[b][g] = hh[g];}}
delete[] hh; delete[] ll;}
////////////////////////////////////////////////
void bi_dwt2(float** fdwt,int s_x,int s_y){
% Decomposition 2-D Wavelet Transform
% x is Image input
% ld is filter cofficient (lowpass)
% hd is filter cofficient (highpass)
% cll is coff Approximation cll
% chl is coff Horizontail Detail chl
% clh is coff Vertical Detail clh
% chh is coff Diagonal Detail chh*/
int b,g,pp;
//%Decomposition
if (s_x >= s_y){pp = s_x;} else{pp = s_y;}
float *hh = new float[pp];
//% rows image
for (b=0 ; b<s_x ; b++)
{for(g=0 ; g<s_y ; g++){hh[g] = fdwt[b][g];}
bi_dwt(hh,s_y);
for(g=0 ; g<s_y ;g++){fdwt[b][g] = hh[g];}}
//% columns image
for (b=0; b<s_y ; b++)
{for(g=0; g<s_x ; g++){hh[g] = fdwt[g][b];}
bi_dwt(hh,s_x);
```

```
for(g = 0; g<s_x  ; g++){fdwt[g][b] = hh[g];}}
delete[] hh;}
////////////////////////////////////////////////
void bi_dwt(float* inx,int size_x){
/*decomposition lowpass*/
float bdl[9] = {(float)bdl0,(float)bdl1,(float)bdl2,(float)bdl3,
(float)bdl4,(float)bdl5,(float)bdl6,(float)bdl7,(float)bdl8};
/*decomposition highpass*/
float bdh[7] = {(float)bdh0,(float)bdh1,(float)bdh2,(float)
bdh3,(float)bdh4,(float)bdh5,(float)bdh6};
int gl=4;    int gh=3;
float *out_l = new float[size_x+8];
float *out_h = new float[size_x+6];
float *o_l = new float[size_x+8];
float *o_h = new float[size_x+6];
refind_data_bi9(out_l,inx,size_x);
refind_data_bi7(out_h,inx,size_x);
conv(o_l,bdl,out_l,9,size_x+8);
int st=8;    int nxt=size_x+8;
down_sampling(o_l,st,nxt,1);
conv(o_h,bdh,out_h,7,size_x+6);
st=6; nxt=size_x+6;
down_sampling(o_h,st,nxt,0);
for(int z=0; z<(size_x/2) ; z++)
{inx[z] = o_l[z]; inx[z+(size_x/2)] = o_h[z];}
delete[] out_l; delete[] out_h; delete[] o_l; delete[] o_h;}
////////////////////////////////////////////////
void bi_idwt(float* inx,int size_x)
{float *out_l = new float[size_x+6];
float *out_h = new float[size_x+8];
/*decomposition lowpass*/
float brh[9] = {(float)brh0,(float)brh1,(float)brh2,(float)
brh3,(float)brh4,(float)brh5,(float)brh6,(float)brh7,(float)
brh8};
/*decomposition highpass*/
float brl[7] = {(float)brl0,(float)brl1,(float)brl2,(float)brl3,
(float)brl4,(float)brl5,(float)brl6};
int gh=4;   int gl=3;
float *l = new float[(size_x/2)];
float *h = new float[(size_x/2)];
for(int t=0 ; t<(size_x/2) ; t++){[t] = inx[t];
h[t] = inx[t+(size_x/2)];}
up_sampling(inx,l,size_x,1);
refind_data_bi7(out_l,inx,size_x);
up_sampling(inx,h,size_x,0);
refind_data_bi9(out_h,inx,size_x);
delete[] l;  delete[] h;
conv_p(brl,out_l,7,size_x+6);
conv_p(brh,out_h,9,size_x+8);
for(t = 0; t<size_x ;t++)
{inx[t] = out_l[t+6] + out_h[t+8];}
delete[] out_l; delete[] out_h;}
////////////////////////////////////////////////
void conv(float* out,float* hn,float* xn,int size_x,int size_h)
{int check;
for(int t=0; t<size_h ; t++){out[t] = 0;
for(int p=0; p<size_x ; p++){check = t-p;
if (check >= 0){out[t] = out[t] + hn[p]*xn[check];}
else{break;}}}}
////////////////////////////////////////////////
void refind_data_bi9(float* out,float* in,int size_x)
{//[fliplr(x(2:gl+1)) x fliplr(x(N-gl:N-1))];
out[0] = in[4]; out[1] = in[3];
out[2] = in[2]; out[3] = in[1];
for(int t=0; t < size_x ; t++) {out[t+4] = in[t];}
out[size_x+4] = in[size_x-2];
out[size_x+5] = in[size_x-3];
out[size_x+6] = in[size_x-4];
out[size_x+7] = in[size_x-5];}
////////////////////////////////////////////////
void refind_data_bi7(float* out,float* in,int size_x)
{out[0] = in[3]; out[1] = in[2]; out[2] = in[1];
for(int t=0; t < size_x ; t++){out[t+3] = in[t];}
out[size_x+3] = in[size_x-2];
out[size_x+4] = in[size_x-3];
out[size_x+5] = in[size_x-4];}
////////////////////////////////////////////////
void up_sampling(float* out,float* in,int size_x,int type)
{//type = 0 -> odd  ,= 1 -> even
int check = type; int count = 0;
for(int t=0 ; t<size_x ;t++)
{if(check==1) {out[t] = in[count];
count  = count + 1;  check  = 0;}
else {out[t] = 0; check  = 1;}}}
```

```
////////////////////////////////////////////////
void down_sampling(float* in,int st,int nxt,int type)
{//type = 0 -> odd ,= 1 -> even
int check;check = type; int count=0;
for(int d=st; d<nxt ; d++)
{if (check==1){in[count]=in[d];
count = count + 1; check = 0;}
else{check = 1;}}}
////////////////////////////////////////////////
void conv_p(float* hn,float* xn,int size_x,int size_h)
{int check,t; float* out_s = new float[size_h];
for(t=0; t<size_h ; t++){out_s[t] = 0;
for(int p=0; p<size_x ; p++) {check = t-p;
if (check >= 0){out_s[t] = out_s[t] + hn[p]*xn[check];}
else{break;}}} for(t=0; t<size_h ; t++) {xn[t] = out_s[t];}
delete[] out_s;}
```

โปรแกรม Encode_SPIHT_main.cpp

```
/*-------------------------------------------------------------*/
#include"SPIHT.h"
void SPIHT_Encode(char* LFC_list_type,long*
countp,char* imout,int bit_end,float** fdwt,int s_x,int
s_y,int level)
{//Initial DATA SPIHT = end_bit (bit) -> imout
//%1)Initialization :
output n = [log2(max(i,j){|ci,j|})]
int n,nl; float num_n; double number; int first_x,first_y;
//limit of first Layer and finit Layer
//first Layer
number    = pow((double)2,(double)level);
//pow(x,y) -> x to y power
nl= (int)number; first_x = s_x/(nl); first_y = s_y/(nl);
float* max_out = new float[4];
dis_max_abs(max_out,fdwt,s_x,s_y);
float max_nn,tii; max_nn = 0;
for (int e = 0 ; e < 4 ; e++)
{ tii = max_out[e];
if (max_nn < tii) { max_nn = tii; } }
n  = log2_floor(max_nn);
number    = pow((double)2,(double)n);
num_n     = (float)number;
long count,count_LSP,count_LIP;
long count_LIS,count_LSP_old;
count = 0; count_LSP = 0; count_LIP = 0;
count_LIS = 0; count_LSP_old = 0;
//%Initial LIP size=[s_x*s_y 3]
int* LSP_x = new int[s_x*s_y];
int* LSP_y = new int[s_x*s_y];
int* LSP_abs = new int[s_x*s_y];
//%Initial LIP size=[s_x*s_y 2]
int* LIP_x = new int[s_x*s_y];
int* LIP_y = new int[s_x*s_y];
int i,j,oi,oj;
for (i=0 ; i < first_x ; i++)
{ for (j=0 ; j < first_y ; j++)
{LIP_x[count_LIP] = i;
LIP_y[count_LIP] = j;
count_LIP = count_LIP + 1;}}
//%Initial LIS size=[s_x*s_y 3]
int* LIS_x = new int[s_x*s_y];
int* LIS_y = new int[s_x*s_y];
int* LIS_type = new int[s_x*s_y];
for ( i=0 ; i < first_x ; i++){
for (j=0 ; j < first_y ; j++)
{//Check odd and even
oi = i%2; oj = j%2;
//Check i
if (oi == 0){ oi = 0; }else{ oi = 1; }
//Check j
if (oj == 0){ oj = 0; }else{ oj = 1; }
if ( (oi+oj)!= 0 ) {
IIS_x[count_LIS] = i; LIS_y[count_LIS] = j;
LIS_type[count_LIS] = 0; //%->set type A
count_LIS = count_LIS +1;}}}
//%1.2 Make it LFC
//size of LFC
int count_LFC; count_LFC = (level*4*3) + 4;
int* LFC_type = new int[count_LFC];
LFC_list(LFC_type,fdwt,s_x,s_y,level);
for(e=0; e <count_LFC ;e++)
{ LFC_list_type[e] = (char)LFC_type[e]; }
for (int step_Quantization = 1;
step_Quantization <= step_quan ; step_Quantization++) {
//%2)Sorting pass:
```

```
//%2.1 for each entry (i,j) in the LIP
for (int t=0; t < count_LIP; t++)
//%2.1.1 output Sn(i,j);
{if (bit_end <= count){break;}
float* output = new float[3];
sn_out(output,LIP_x,LIP_y,t,fdwt,num_n);
imout[count] = (int)output[0]; count = count + 1;
//%2.1.2) Sn(i,j) = 1 then move (i,j) to the LSP and output
the sign of cij
if ((int)output[0] == 1)
{LSP_x[count_LSP]  = LIP_x[t];
 LSP_y[count_LSP]  = LIP_y[t];
LSP_abs[count_LSP] = (int)output[1];
count_LSP = count_LSP+1;
//%Output the sign of cij
if (output[2] != 0){ imout[count] = 1; }
else{ imout[count] = 0;}
//%END of Output the sign of cij
count = count + 1;
LIP_x[t] = -1; LIP_y[t] = -1; }
//%End of the Sn(i,j)=1
if (bit_end <= count){break;}
}//%END of is Sn(i,j) = then move (i,j) to the LSP
//%2.1.2) Move LIP(i,j) at sn(i,j)=1
long* count_LIP_new = new long[1];
cut_LIP(count_LIP_new,LIP_x,LIP_y,count_LIP);
count_LIP = count_LIP_new[0];
//% END of Loop for each entry (i,j) in the LIP
//%2.2 for each entry (i,j) in the LIS
//%Initial parameter 2.2
long count_LIS_old;
count_LIS_old = count_LIS;
int change,set_add,ccp;
change = 100;
set_add = 0;  ccp = 0;
for (long LIS_length = 0 ; LIS_length < 100000000 ;
LIS_length ++){
if ((change == 0) && (count_LIS_old <= LIS_length))
{break;}
if (LIS_length==0)
{change = 0;}
if ((count_LIS_old <= LIS_length) && set_add==1)
{ change = change - 1 ; }
//%Find Do(i,j) , LO(i,j)
i = LIS_x[LIS_length]; j = LIS_y[LIS_length];
//%Check LFC list
int LFC_D = 0; //%LFC O(i,j)  int LFC_L = 0;
//%LFC L(i,j)
//O_list
int **DOij_list = new int*[4];
for(int  ii=0; ii<4; ii++)
{DOij_list[ii] = new int[2];}
int LFC_x,LFC_y,Let_ijk=0;
int* count_set = new int[1];
count_set[0] = 4;
coor_offspring_s(count_set,DOij_list,i,j,s_x,s_y,level);
LFC_x = DOij_list[0][0]; LFC_y = DOij_list[0][1];
int type_LFC_label;
type_LFC_label =type_label(LFC_x,LFC_y,s_x,s_y,level);
if (LFC_type[type_LFC_label] <= step_Quantization)
{LFC_D = 1;  //%->No LFC D(i,j)}
//O_listo =
coor_offspring(LFC_x,LFC_y,x_pic,y_pic,level);
int **O_listo = new int*[4];
for(ii=0; ii<4; ii++)
{O_listo[ii] = new int[2];}
////////////////////////////////////////////////
coor_offspring(O_listo,LFC_x,LFC_y,s_x,s_y,level);
int LFC_ox,LFC_oy;
if (O_listo[0][0] != -1){LFC_ox = O_listo[0][0];
LFC_oy = O_listo[0][1];
type_LFC_label =
type_label(LFC_ox,LFC_oy,s_x,s_y,level);
if (LFC_type[type_LFC_label] <= step_Quantization)
{ LFC_L = 1; //%->No LFC L(i,j) }}
int check_work = 0; int value_SD,value_SL,Lij_set,yp;
int *type_ij = new int[3];
if ((LFC_D + LFC_L) != 0)
{coor_descendants_s(type_ij,i,j,num_n,fdwt,s_x,s_y,level);
value_SD = type_ij[0];
value_SL = type_ij[1];
Lij_set  = type_ij[2];
check_work = 1;}
float absx,d_fdwt; int check_LIP = 0;
```

```
if (check_work == 1) //%2.2.1 if the entry is of type A
{if (LIS_type[LIS_length] == 0) //%Output Sn(D(i,j));{
if (bit_end <= count){goto a;}
imout[count] = value_SD;
count = count + 1;
if (value_SD == 1)
{for (int OD_A=0 ; OD_A < count_set[0] ; OD_A++)
//%Output Sn(OD(k,l))
{//DATA DOij_list
d_fdwt = fdwt[DOij_list[OD_A][0]][DOij_list[OD_A][1]];
if (d_fdwt < 0){absx = d_fdwt * -1;}
else{absx = d_fdwt;}
if (absx >= num_n)
//%Sn(O(i,j)) is significant
{imout[count] = 1;
count = count + 1;
LSP_x[count_LSP] = DOij_list[OD_A][0];
LSP_y[count_LSP] = DOij_list[OD_A][1];
LSP_abs[count_LSP] = (int)(absx - num_n);
count_LSP = count_LSP+1;
//%Output sing of c(k,l)
if (d_fdwt >= 0) {imout[count] = 1;}
else{imout[count] = 0;}
count = count + 1;
if (bit_end <= count){goto a;}}
else{imout[count] = 0;
count = count + 1;
LIP_x[count_LIP] = DOij_list[OD_A][0];
LIP_y[count_LIP] = DOij_list[OD_A][1];
count_LIP = count_LIP + 1;
check_LIP = check_LIP + 1;
if (bit_end <= count){goto a;}}}
if (Lij_set == 1)
{if (check_LIP == count_set[0])
{for(yp = 0 ; yp < count_set[0] ; yp++)
{LIS_x[count_LIS]  = DOij_list[yp][0];
 LIS_y[count_LIS]  = DOij_list[yp][1];
LIS_type[count_LIS] = 0;
count_LIS = count_LIS + 1;
change    = change + 1;} set_add = 1;
//%remove (i,j) from the LIS_B
LIS_x[LIS_length] = -1;
LIS_y[LIS_length] = -1;
LIS_type[LIS_length] = 0; }
else{if (LFC_L == 1)//%Output Sn(L(i,j))
{if (bit_end <= count){goto a;}
imout[count] = value_SL;
count = count + 1;
if (value_SL == 1)
{for(int yp = 0 ; yp < count_set[0] ; yp++)
{ LIS_x[count_LIS]  = DOij_list[yp][0];
LIS_y[count_LIS]  = DOij_list[yp][1];
LIS_type[count_LIS] = 0;
count_LIS = count_LIS + 1;
change=change + 1;} //%remove (i,j) from the LIS_B
LIS_x[LIS_length]  = -1;
LIS_y[LIS_length]  = -1;
LIS_type[LIS_length] = 1;
set_add = 1;}
else{LIS_type[LIS_length] = 1;}
if (bit_end <= count){goto a;}}
else{LIS_type[LIS_length] = 1;}}}
else{LIS_x[LIS_length] = -1;
LIS_y[LIS_length] = -1;
LIS_type[LIS_length] = 0; }}}
//%2.2.2 if the entry is of type B
else{if (LFC_L == 1)//%Output Sn(L(i,j))
{if (bit_end <= count){goto a;}
imout[count] = value_SL;
count = count + 1;
if (value_SL == 1)
{/for(int yp = 0 ; yp < count_set[0] ; yp++)
{ LIS_x[count_LIS]  = DOij_list[yp][0];
LIS_y[count_LIS]  = DOij_list[yp][1];
LIS_type[count_LIS] = 0;
count_LIS = count_LIS + 1;
change=change + 1;}
//%remove (i,j) from the LIS_B
LIS_x[LIS_length]  = -1;
LIS_y[LIS_length]  = -1;
LIS_type[LIS_length] = 1;
set_add = 1;}
if (bit_end <= count){goto a;}}}}}
//%remove (i,j)
```

```
long* count_LIS_new = new long[1];
cut_LIS(count_LIS_new,LIS_x,LIS_y,LIS_type,count_LIS)
; count_LIS = count_LIS_new[0];
//%3 Refinement Pass: for each entry (i,j) in the LSP
long* count_new = new long[1];
refinement_pass(count_new,imout,LSP_abs,count_LSP,
(num_n/2),bit_end,count);
count = count_new[0]; num_n = num_n/2;}
a: countp[0] = count;
delete[] LSP_x; delete[] LSP_y;
delete[] LSP_abs; delete[] LIP_x;
delete[] LIP_y;  delete[] LIS_x;
delete[] LIS_y; delete[] LIS_type;}
```

โปรแกรม Encode_SPIHT_function.cpp

```
/*-------------------------------------------------------------*/
#include"SPIHT.h"
void coor_descendants_s(int* type_ij,int i,int j,float
threshold,float** fdwt,int s_x,int s_y,int level)
{/*Input
%i,j = coordinates
%thres = threshold
%Output
%Value_SD = output Sn(D(i,j));
% = 0 -> insignificant D(i,j)
% = 1 -> significant  D(i,j)
%value_SL = output Sn(L(i,j));
%          = 0 -> insignificant L(i,j)
%          = 1 -> significant  L(i,j)
%Lij_set  = set of L(i,j)
%= 0 -> No membership
%= 1 -> Yes membership
%type_ij
type_ij[0] = value_SD;
type_ij[1] = value_SL;
type_ij[2] = Lij_set; */
//Initial parameter
int Lij_set  = 0; int value_SD = 0;
int value_SL = 0; int null = 1;
int *countOO = new int[1];
float opp;
/*%X-> i %Y-> j
%limit of first Layer and finit Layer
%first Layer */
double number;
int p,first_x,first_y,finit_x,finit_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p);  first_y = s_y/(p);
//%finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//%Group 1 ->DWT Level 1 ,LH1 ,HL1 ,HH1
if ((i>= finit_x && i<s_x && j>=0 && j<s_y) ||
(i>=0 && i<finit_x && j>=finit_y && j<s_y))
{Lij_set = 0;null = 0; }
if (null != 0){
//O_list
int **O_list = new int*[4];
for(int  ii=0; ii<4; ii++){O_list[ii] = new int[2];}
int ss_i,ss_j;
//%First Step search
coor_offspring_s(countOO,O_list,i,j,s_x,s_y,level);
//O_list_new
int *O_list_i = new int[350];
int *O_list_j = new int[350];
//display(O_list,4,2);
for (int n=0 ; n < countOO[0] ; n++)
{//Lij_set  = 1;
ss_i = O_list[n][0]; ss_j = O_list[n][1];
//%Sn(D(i,j))
if (fdwt[ss_i][ss_j] < 0)
{opp = fdwt[ss_i][ss_j] * -1;}
else{opp = fdwt[ss_i][ss_j];}
if ( opp >= threshold){ value_SD = 1;}
O_list_i[n] = O_list[n][0];
O_list_j[n] = O_list[n][1];}
int count_oln,count_oln_old,end_count;
count_oln = countOO[0];  count_oln_old = count_oln;
//Next Step search
if (O_list[0][0] != -1){
for (int pso=0 ; pso < count_oln_old ; pso ++)
{ss_i = O_list_i[pso];
```

```
ss_j = O_list_j[pso];
coor_offspring(O_list,ss_i,ss_j,s_x,s_y,level);
if (O_list[0][0] != -1){Lij_set = 1; null = 1;
for (n=0 ; n < 4 ; n++){
ss_i = O_list[n][0]; ss_j = O_list[n][1];
O_list_i[count_oln] = ss_i;
O_list_j[count_oln] = ss_j;
count_oln = count_oln + 1;
//%Sn(L(i,j)
if (fdwt[ss_i][ss_j] < 0)
{opp = fdwt[ss_i][ss_j] * -1;}
else{opp = fdwt[ss_i][ss_j];}
if ( opp >= threshold){
value_SD = 1; value_SL = 1;
null = 0;goto a;}}}}}
int point=2;
if (null == 1){
for (int ttt = 1; ttt <= 100 ; ttt++){
null = 0; end_count = count_oln;
for (int psp = count_oln_old ; psp < end_count ;psp ++)
{ ss_i = O_list_i[psp];
ss_j = O_list_j[psp];
coor_offspring(O_list,ss_i,ss_j,s_x,s_y,level);
if (O_list[0][0] != -1){ Lij_set = 1; null = 1;
for (n=0 ; n < 4 ; n++){if (point <= 3)
{ss_i = O_list[n][0];
ss_j = O_list[n][1];
O_list_i[count_oln] = ss_i;
O_list_j[count_oln] = ss_j;
count_oln = count_oln + 1;}
//%Sn(L(i,j)
if (fdwt[ss_i][ss_j] < 0)
{opp = fdwt[ss_i][ss_j] * -1;}
else{opp = fdwt[ss_i][ss_j];}
if ( opp >= threshold){
value_SD = 1; value_SL = 1; null = 0;
goto a; }}}}
count_oln_old = end_count;
point = point + 1;
if (null != 1){goto a;}}
delete[] O_list_i;
delete[] O_list_j; } }
a: type_ij[0] = value_SD;
type_ij[1] = value_SL; type_ij[2] = Lij_set;}
//////////////////////////////////////////////////
void coor_offspring_s(int* countO,int** O_list,int i,int j,int
s_x,int s_y,int level)
{ i = i + 1; j = j + 1;
double number;
int count_O = 0;
int p,first_x,first_y,finit_x,finit_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p); first_y = s_y/(p);
//finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//Find odd and even
//output = 1 -> odd
//output = 0 -> even
int oi,oj;
//Check i
oi = i%2; oj = j%2;
if (oi == 0){oi = 0;} else{oi = 1;}
//Check j
if (oj == 0){oj = 0;} else{oj = 1;}
int Check = 1;
//No root if ->Yes L(i,j)
//Level END
int px,py;
if (i>=1 && i<=first_x && j>=1 && j<=first_y)
{if ((oi+oj)!=2){ px=first_x-1;
py=first_y-1;
if (oi==1 && oj==0)
{O_list[0][0] = i; O_list[0][1] = j+py;}
if (oi==0 && oj==1)
{O_list[0][0] = i+px; O_list[0][1] = j; }
if (oi==0 && oj==0)
{O_list[0][0] = i+px; O_list[0][1] = j+py;}
Check = 0;}
else{ count_O = 0;
O_list[count_O][0] = i+1-1;
O_list[count_O][1] = j-1;
```

```
//%1)
O_list[count_O+1][0] = i-1;
O_list[count_O+1][1] = j+1-1;
//%2)
O_list[count_O+2][0] = i+1-1;
O_list[count_O+2][1] = j-1+1;
count_O = 3; Check = 2; } }
//NO root if -> No L(i,j)
//Level 1 (LH1 HL1 HH1)
int a_1,a_2;
a_1 = (finit_x+1); a_2 = (finit_y+1);
if ((i>= a_1 && i<=s_x && j>=1 && j<=s_y) ||
(i>=1 && i<=finit_x && j>= a_2 && j<=s_y) )
{O_list[0][0] = -1;      //(-1)
O_list[0][1] = -1;       //(-1)
Check = 2;}
//Root of O(i,j) ->Yes L(i,j)
if (Check == 1){
O_list[0][0]=(int)((i*2)-1);
O_list[0][1]=(int)((j*2)-1);}
if (Check !=2){
O_list[0][0]-=1; O_list[0][1]-=1;
//Find root of O(i,j)
O_list[1][0] = O_list[0][0];
O_list[1][1] = O_list[0][1] + 1;
O_list[2][0] = O_list[0][0] + 1;
O_list[2][1] = O_list[0][1];
O_list[3][0] = O_list[0][0] + 1;
O_list[3][1] = O_list[0][1] + 1;
count_O = 4;} countO[0] = count_O;}
////////////////////////////////////////////////
int find_max_coff(float** fdwt,int s_x,int s_y,int level)
{int first_x,first_y,n,nl; float max_nn,tii;
double number;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
nl = (int)number;
first_x = s_x/(nl); first_y = s_y/(nl);
//max(max(abs(pic)))
float* max_out = new float[4];
dis_max_abs(max_out,fdwt,s_x,s_y);
max_nn = 0;
for (int e = 0 ; e < 4 ; e++){ tii = max_out[e];
if (max_nn < tii){max_nn = tii;}}
n  = log2_floor(max_nn);
delete[] max_out;
return(n); }
////////////////////////////////////////////////
void coor_descendants(int* type_ij,int i,int j,float
threshold,float** fdwt,int s_x,int s_y,int level)
{/*Input
%i,j = coordinates
%thres = threshold
%Output
%Value_SD = output Sn(D(i,j));
%              = 0 -> insignificant D(i,j)
%              = 1 -> significant   D(i,j)
%value_SL = output Sn(L(i,j));
%              = 0 -> insignificant L(i,j)
%              = 1 -> significant   L(i,j)
%Lij_set  = set of L(i,j)
%              = 0 -> No membership
%             = 1 -> Yes membership
%type_ij
type_ij[0] = value_SD;  type_ij[1] = value_SL;
type_ij[2] = Lij_set; */
//Initial parameter
int Lij_set  = 0;  int value_SD = 0;
int value_SL = 0; int null = 1;
float opp;
%limit of first Layer and finit Layer
%first Layer */
double number;
int  p,first_x,first_y,finit_x,finit_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p); first_y = s_y/(p);
//%finit Layer
finit_x = s_x/2;   finit_y = s_y/2;
//%Group 1 ->DWT Level 1 ,LH1 ,HL1 ,HH1
```

```
if ((i>= finit_x && i<s_x && j>=0 && j<s_y) || (i>=0 && i
<finit_x && j>=finit_y && j<s_y))
{Lij_set = 0;null = 0;}
if (null != 0){
//O_list
int **O_list = new int*[4];
for(int  ii=0; ii<4; ii++)
{O_list[ii] = new int[2];}
int ss_i,ss_j;
//%First Step search
coor_offspring(O_list,i,j,s_x,s_y,level);
for (int n=0 ; n < 4 ; n++){
ss_i = O_list[n][0]; ss_j = O_list[n][1];
//%Sn(D(i,j))
if (fdwt[ss_i][ss_j] < 0)
{opp = fdwt[ss_i][ss_j] * -1;}
else{opp = fdwt[ss_i][ss_j];}
if ( opp >= threshold){ value_SD = 1; }}
//O_list_new
int *O_list_i = new int[350];
int *O_list_j = new int[350];
int count_oln,count_oln_old,end_count;
O_list_i[0] = O_list[0][0];
O_list_j[0] = O_list[0][1];
O_list_i[1] = O_list[1][0];
O_list_j[1] = O_list[1][1];
O_list_i[2] = O_list[2][0];
O_list_j[2] = O_list[2][1];
O_list_i[3] = O_list[3][0];
O_list_j[3] = O_list[3][1];
count_oln = 4;
count_oln_old = count_oln;
//Next Step search
if (O_list[0][0] != -1){ Lij_set  = 1;
for (int pso=0 ; pso < 4 ; pso ++){
ss_i = O_list_i[pso]; ss_j = O_list_j[pso];
coor_offspring(O_list,ss_i,ss_j,s_x,s_y,level);
if (O_list[0][0] != -1){null = 1;
for (n=0 ; n < 4 ; n++){
ss_i = O_list[n][0];  ss_j = O_list[n][1];
O_list_i[count_oln] = ss_i; O_list_j[count_oln] = ss_j;
count_oln = count_oln + 1;
//%Sn(L(i,j)
if (fdwt[ss_i][ss_j] < 0)
{opp = fdwt[ss_i][ss_j] * -1;}
else{opp = fdwt[ss_i][ss_j];}
if ( opp >= threshold){
value_SD = 1;  value_SL = 1;
null = 0; goto a;}}}}}
int point=2;
if (null == 1){
for (int ttt = 1; ttt <= 100 ; ttt++){
Lij_set  = 1; null = 0;  end_count = count_oln;
for (int psp = count_oln_old ; psp < end_count ;psp ++)
{ss_i = O_list_i[psp];
ss_j = O_list_j[psp];
coor_offspring(O_list,ss_i,ss_j,s_x,s_y,level);
if (O_list[0][0] != -1){null = 1;
for (n=0 ; n < 4 ; n++){
if (point <= 3){
ss_i = O_list[n][0];  ss_j = O_list[n][1];
O_list_i[count_oln] = ss_i;
O_list_j[count_oln] = ss_j;
count_oln = count_oln + 1;}//%Sn(L(i,j)
if (fdwt[ss_i][ss_j] < 0)
{opp = fdwt[ss_i][ss_j] * -1;}
else{opp = fdwt[ss_i][ss_j];}
if ( opp >= threshold){ value_SD = 1;
value_SL = 1;  null = 0;
goto a; } } } }
count_oln_old = end_count; point = point + 1;
if (null != 1)
{goto a;} }
delete[] O_list_i;
delete[] O_list_j; } }
a: type_ij[0] = value_SD;
type_ij[1] = value_SL;
type_ij[2] = Lij_set; }
////////////////////////////////////////////////
void coor_offspring(int** O_list,int i,int j,int s_x,int s_y,int
level)
{ i = i + 1; j = j + 1;
double number;
int p,first_x,first_y,finit_x,finit_y;
```

```
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p);  first_y = s_y/(p);
//finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//Find odd and even
//output = 1 -> odd output = 0 -> even
int oi,oj;
//Check i
oi = i%2; oj = j%2;
if (oi == 0){oi = 0;}
else{oi = 1;}
//Check j
if (oj == 0){oj = 0;}
else{oj = 1;}
int Check = 1;
//No root if ->Yes L(i,j)
//Level END
int px,py;
if (i>=1 && i<=first_x && j>=1 && j<=first_y)
{if ((oi+oj)!=2){px=first_x-1; py=first_y-1;
if (oi==1 && oj==0){ O_list[0][0] = i;
O_list[0][1] = j+py; }
if (oi==0 && oj==1){O_list[0][0] = i+px;
O_list[0][1] = j; }
if (oi==0 && oj==0) {O_list[0][0] = i+px;
O_list[0][1] = j+py;}
Check = 0;}
else{ O_list[0][0] = -1;
O_list[0][1] = -1; Check = 2; }}
//NO root if -> No L(i,j)
//Level 1 (LH1 HL1 HH1)
int a_1,a_2;
a_1 = (finit_x+1);
a_2 = (finit_y+1);
if ((i>= a_1 && i<=s_x && j>=1 && j<=s_y) ||
(i>=1 && i<=finit_x && j>= a_2 && j<=s_y) )
{O_list[0][0] = -1;     //(-1)
O_list[0][1] = -1;      //(-1)
Check = 2;}
//Root of O(i,j) ->Yes L(i,j)
if (Check == 1){
O_list[0][0]=(int)((i*2)-1);
O_list[0][1]=(int)((j*2)-1);}
if (Check !=2){
O_list[0][0]-=1; O_list[0][1]-=1;
//Find root of O(i,j)
O_list[1][0] = O_list[0][0];
O_list[1][1] = O_list[0][1] + 1;
O_list[2][0] = O_list[0][0] + 1;
O_list[2][1] = O_list[0][1];
O_list[3][0] = O_list[0][0] + 1;
O_list[3][1] = O_list[0][1] + 1;}}
////////////////////////////////////////////////
void refinement_pass(long* count_new,char* imout,int*
LSP_abs,long count_LSP,float num_n,long bit_end,long
count)
{float ans;
for (int t=0 ; t < count_LSP ; t++){
if (bit_end <= count){break;}
ans = (float)LSP_abs[t];
if ((ans >= num_n) && (ans != 0)){
imout[count] = 1;
LSP_abs[t]  = (int)(ans - num_n);}
else{imout[count] = 0;}
count = count + 1;
if (bit_end <= count){break;}}
count_new[0] = count;}
////////////////////////////////////////////////
void cut_LIS(long* count_LIS_new,int* LIS_x,int*
LIS_y,int* LIS_type,long count_LIS)
{ long count=0;
for(long z=0 ; z < count_LIS ; z++){
if(LIS_x[z]!=-1){count = count + 1;}}
int* x_new = new int[count];
int* y_new = new int[count];
int* z_new = new int[count];
count = 0;
for( z=0 ; z < count_LIS ; z++)
{if(LIS_x[z]!=-1)
{x_new[count] = LIS_x[z]; y_new[count] = LIS_y[z];
z_new[count] = LIS_type[z];
```

```
LIS_x[z]    = -1;  count = count + 1;}}
for( z=0 ; z < count ; z++){
LIS_x[z]    = x_new[z];  LIS_y[z]    = y_new[z];
LIS_type[z]  = z_new[z]; }
count_LIS_new[0] = count;
delete[] x_new;  delete[] y_new;
delete[] z_new; }
////////////////////////////////////////////////
void cut_LIP(long* count_LIP_new,int* LIP_x,int*
LIP_y,long count_LIP)
{ long count=0;
for(long z=0 ; z < count_LIP ; z++){
if(LIP_x[z]!=-1){count = count + 1;}}
int* x_new = new int[count];
int* y_new = new int[count];
count = 0;
for( z=0 ; z < count_LIP ; z++){
if(LIP_x[z]!=-1){
x_new[count] = LIP_x[z];
y_new[count] = LIP_y[z];
LIP_x[z]    = -1;
count = count + 1; }}
for( z=0 ; z < count ; z++)
{LIP_x[z] = x_new[z];
LIP_y[z]  = y_new[z];}
count_LIP_new[0] = count;
delete[] x_new; delete[] y_new;}
////////////////////////////////////////////////
void sn_out(float* output,int* LIP_x,int* LIP_y,int t,float**
fdwt,float num_n)
{/* output[0] = significant   -> 1
    No significant -> 0
    output[1] = abs(data(i,j)) - num_n
    output[2] = sign     -> 1 -> (+)
                         -> 0 -> (-) */
 //%coordinates Sn(i,j)
 int i,j;
 i =  LIP_x[t]; j =  LIP_y[t];
 //%Output
 float data; data = fdwt[i][j];
 //Check sign and abs(data)
 if (data < 0){output[2] = 0; // (-)
data = data * -1;}
else{output[2] = 1; // (+)}
//Check significant pixel and save data
if (data >= num_n)
{output[0] = 1;//significant}
else{output[0] = 0;     //no significant}
data = data - num_n;
output[1] = data;  // abs(data) - num_n}
```

โปรแกรม LFC.cpp

```
/*-------------------------------------------------------*/
#include"SPIHT.h"
////////////////////////////////////////////////
//LFC list
////////////////////////////////////////////////
int type_label(int x,int y,int s_x,int s_y,int level)
{ //%X-> x //%Y-> y
x = x + 1; y = y + 1;
//%limit of first Layer and finit Layer
//%first Layer
double number; int  p,first_x,first_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p); first_y = s_y/(p);
int tx,ty,label;
//%------------------------------------ level - 1
//%(LL)1-4
if ((x<=first_x) && (y<=first_y)){
tx = first_x/2; ty = first_y/2;
if ((x<=tx) && (y<=ty)){label = 1;}
if ((x<=tx) && (y>ty)){label = 2;}
if ((x>tx) && (y<=ty)){label = 3;}
if ((x>tx) && (y>ty)){label = 4;}}
int ll,a,c,delta;
for (int l=0 ; l <= (level-1) ; l++)
{delta = 12*l; //ll = (2^l);
number = pow((double)2,(double)l);
ll= (int)number;  a = (first_x*ll); c = (first_x*2*ll);
//%------------------------------------ layer
//%(HL)
```

```
if ((x<=a) && (y>a && y<=c))
{tx = a/2; ty = (a+c)/2;
if ((x<=tx) && (y<=ty))
{label = 5+delta; break; }
if ((x<=tx) && (y>ty))
{label = 6+delta; break;}
if ((x>tx) && (y<=ty))
{label = 7+delta; break; }
if ((x>tx) && (y>ty))
{label = 8+delta; break; } }
//%(LH)
if ((x>a && x<=c) && (y<=a))
{tx = (a+c)/2; ty = a/2;
if ((x<=tx) && (y<=ty))
{ label = 9+delta; break;}
if ((x<=tx) && (y>ty))
{label = 10+delta; break;
if ((x>tx) && (y<=ty))
{label = 11+delta; break; }
if ((x>tx) && (y>ty))
{label = 12+delta; break; }
//%(HH)
if ((x>a && x<=c) && (y>a && y<=c))
{tx = (a+c)/2; ty = (a+c)/2;
if ((x<=tx) && (y<=ty))
{label = 13+delta; break; }
if ((x<=tx) && (y>ty))
{ label = 14+delta; break; }
if ((x>tx) && (y<=ty))
{label = 15+delta; break;}
if ((x>tx) && (y>ty))
{label = 16+delta;break;}}}
label = label - 1; return(label);}
//////////////////////////////////////////////////
void LFC_list(int* q_LFC,float** image_in,int s_x,int
s_y,int level)
{//%limit of first Layer and finit Layer
//%first Layer
double number;
int p,first_x,first_y;
//limit of first Layer and finit Layer
//first Layer
number    = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p); first_y = s_y/(p);
//size of LFC list
int count_LFC,x1,x2,y1,y2;
count_LFC = (level*4*3) + 4;
float* LFC = new float[count_LFC];
count_LFC  = 0;
//%LL
//cll = image_in(1:first_x,1:first_y);
float** output = new float*[s_x/2];
for(int  ii=0; ii< (s_x/2); ii++)
{output[ii] = new float[s_y/2];}
x1 = 0; y1 = 0;
x2 = first_x - 1;
y2 = first_y - 1;
cut_image(output,image_in,0,x2,0,y2);
float* max_out = new float[4];
dis_max_abs(max_out,output,first_x,first_y);
for (int n=0 ; n < 4 ; n++)
{LFC[count_LFC] = max_out[n];
count_LFC  = count_LFC + 1;  }
int  l,a,b,c;
//%Layer 1-level
for (int degree=0; degree <= (level-1) ; degree++)
{number    = pow((double)2,(double)degree);
l = (int)number;
a = (first_x*l) - 1;
b = (first_x*l+1) - 1;
c = (first_x*2*l) - 1;
//%HL
cut_image(output,image_in,0,a,b,c);
x1 = a + 1;  y1 = c - b +1;
dis_max_abs(max_out,output,x1,y1);
for (n=0 ; n < 4 ; n++)
{LFC[count_LFC] = max_out[n];
count_LFC = count_LFC + 1;  }
//%LH
cut_image(output,image_in,b,c,0,a);
x1 = c - b + 1; y1 = a;
dis_max_abs(max_out,output,x1,y1);
for (n=0 ; n < 4 ; n++)
```

```
{LFC[count_LFC] = max_out[n];
count_LFC  = count_LFC + 1;  }
//%HH
cut_image(output,image_in,b,c,b,c);
x1 = c - b + 1;  y1 = x1;
dis_max_abs(max_out,output,x1,y1);
for (n=0 ; n < 4 ; n++)
{LFC[count_LFC] = max_out[n];
count_LFC     = count_LFC + 1;  } }
int type_ii,max_n; float tii,max_nn; max_nn = 0;
for (int e = 0 ; e < 4 ; e++){tii = LFC[e];
if (max_nn < tii){max_nn = tii;} }
max_n  = log2_floor(max_nn);
for (e = 0 ; e < count_LFC ; e++){ tii = LFC[e];
type_ii = floor_LFC_n(tii,max_n);
q_LFC[e] = type_ii; } }
////////////////////////////////////////////////
int floor_LFC_n(float max_out_n,int max_n)
{ int out,output;
out = log2_floor(max_out_n);
output = max_n - out + 1 ;
return(output);}
////////////////////////////////////////////////
int log2_floor(float data)
{double x,y; int out;
x = (double)data; y = log(x);
x = log(2.0); out = (int)(y/x); return(out);}
////////////////////////////////////////////////
void cut_image(float** output,float** input,int x_start,int
x_end,int y_start,int y_end)
{ int s_i_x,s_i_y; s_i_x = 0;
for (int index_row=x_start;index_row<=x_end;
Index_row++) {s_i_y = 0;
for (int index_col=y_start;index_col<=y_end;index_col++)
{output[s_i_x][s_i_y] = input[index_row][index_col];
s_i_y = s_i_y + 1; }
s_i_x = s_i_x + 1; } }
////////////////////////////////////////////////
void dis_max_abs(float* max_out,float** input,int ss_x,int
ss_y)
{ int x,y,il,jl;
float l1,l2,l3,l4,s_data;
l1 = 0; l2 = 0; l3 = 0; l4 = 0;
//l1
x = ss_x/2; y = ss_y/2;
for(il = 0; il < x; il++) { for(jl = 0; jl < y; jl++){
if (input[il][jl] < 0){s_data = input[il][jl] * -1;}
else{s_data = input[il][jl];}
if ( l1 < s_data){l1 = s_data;} }}
//l2
for(il = 0; il < x; il++){ for(jl = y; jl < ss_y; jl++){
if (input[il][jl] < 0){ s_data = input[il][jl] * -1;}
else{s_data = input[il][jl];}
if ( l2 < s_data){l2 = s_data;} } }
//l3
for(il = x; il < ss_x; il++){ for(jl = 0; jl < y; jl++){
if (input[il][jl] < 0){s_data = input[il][jl] * -1;}
else{s_data = input[il][jl];}
if ( l3 < s_data){l3 = s_data;} } }
//l4
for(il = x; il < ss_x; il++){ for(jl = y; jl < ss_y; jl++)
{ if (input[il][jl] < 0)
{s_data = input[il][jl] * -1;}
else{s_data = input[il][jl];}
if ( l4 < s_data){l4 = s_data;} } }
max_out[0] = l1;  max_out[1] = l2;
max_out[2] = l3; max_out[3] = l4;}
////////////////////////////////////////////////
void round_image(float** input,int s_x,int s_y)
{int store_int;
for(int i = 0 ; i <s_x  ;i++){for(int j = 0 ; j<s_y ;j++)
{ if (input[i][j] >= 0)
{store_int = (int)(input[i][j]+0.5);
input[i][j] = (float)store_int;}
else{store_int = (int)(input[i][j]-0.5);
input[i][j] = (float)store_int;}}}}
```

ภาคผนวก ข

รายละเอียดโปรแกรมการถอดรหัสอัลกอริทึม SPIHT ที่ผ่านการปรับปรุง

โปรแกรม SPIHT.h

```
/*-------------------------------------------------------*/
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include"constant.h"
#include<iostream.h>
#define max_nr -1     //Check No root of o(i,j)
//DWT and IDWT (1D and 2D) -> bi9-7
void conv(float* out,float* hn,float* xn,int size_h,int
size_x);
void conv_p(float* hn,float* xn,int size_x,int size_h);
//DWT
void refind_data_bi9(float* out,float* in,int size_x);
void refind_data_bi7(float* out,float* in,int size_x);
void bi_dwt(float* inx,int size_x);
void down_sampling(float* in,int st,int nxt,int type);
void bi_dwt2(float** fdwt,int s_x,int s_y);
//IDWT
void bi_idwt(float* inx,int size_x);
void up_sampling(float* out,float* in,int size_x,int type);
void bi_idwt2(float** fdwt,int s_x,int s_y);
//Encode SPIHT
void coor_offspring(int** O_list,int i,int j,int s_x,int s_y,int
level);
void coor_descendants(int*,int i,int j,float threshold,float**
fdwt,int s_x,int s_y,int level);
//LFC SPIHT
void LFC_list(int* q_LFC,float** image_in,int s_x,int
s_y,int level);
void cut_image(float** output,float** input,int x_start,int
x_end,int y_start,int y_end);
void dis_max_abs(float* max_out,float** input,int ss_x,int
ss_y);
int type_label(int x,int y,int s_x,int s_y,int level);
int floor_LFC_n(float max_out_n,int max_n);
int log2_floor(float data);
void sn_out(float* output,int* LIP_x,int* LIP_y,int t,float**
fdwt,float num_n);
void cut_LIP(long* count_LIP_new,int* LIP_x,int*
LIP_y,long count_LIP);
void cut_LIS(long* count_LIS_new,int* LIS_x,int*
LIS_y,int* LIS_type,long count_LIS);
void refinement_pass(long* count_new,char* imout,int*
LSP_abs,long count_LSP,float num_n,long bit_end,long
count);
int find_max_coff(float** fdwt,int s_x,int s_y,int level);
//DECODE SPIHT
void SPIHT_Decode(char* LFC_list_type,float**
output_image,char* imout,int bit_end,int s_x,int s_y,int
level,int n);
void i_coor_descendants(int** O_list,int* type_ij,int i,int
j,int s_x,int s_y,int level);
void i_refinement_pass(long* count_new,char* imout,int*
LSP_abs,long count_LSP,float num_n,long bit_end,long
count);
void re_image(int** output_image,int* LSP_x,int*
LSP_y,int* LSP_abs,int* LSP_sign,long count_LSP);
void coor_descendants_s(int* type_ij,int i,int j,float
threshold,float** fdwt,int s_x,int s_y,int level);
void coor_offspring_s(int* countO,int** O_list,int i,int j,int
s_x,int s_y,int level);
void i_coor_descendants_s(int *count_set,int** O_list,int*
type_ij,int i,int j,int s_x,int s_y,int level);
void round_image_pass(float** input,int s_x,int s_y,float
bitrate);
void round_image_abs(float** input,int s_x,int s_y);
```

โปรแกรม coff.h

เหมือนในภาคผนวก ก.

โปรแกรม main.cpp

```
/*-------------------------------------------------------*/
#include"SPIHT.h"
#include"coff.h"
main(){
////////////////////////////////////////////////////////////////
//Load file Header SPIHT
int bit_end,in1[1];     //Size Data
int s_x,in2[1];         //->Dimension X
int s_y,in3[1];         //->Dimension Y
int level,in4[1];       //-> level wavelet tranfrom
```

```
int n,in5[1];              //-> Initial n max(coff)
////////////////////////////////////////////////
FILE *pheader;
pheader = fopen("header.sim","rb");
fread(in1,sizeof(int),1,pheader);  //size of data wavelet
SPIHT
fread(in2,sizeof(int),1,pheader); //hight image
fread(in3,sizeof(int),1,pheader); //wide image
fread(in4,sizeof(int),1,pheader); //level decomposition
fread(in5,sizeof(int),1,pheader); //number step Quantization
bit_end = in1[0];
s_x = in2[0];  s_y = in3[0];  level = in4[0];  n = in5[0];
//LFC List
int count_LFC; count_LFC = (level*4*3) + 4;
char* LFC_list_type = new char[count_LFC];
fread(LFC_list_type,sizeof(char),count_LFC,pheader);
fclose(pheader);
////////////////////////////////////////////////
//Load file Data SPIHT
char* imout = new char[bit_end];
FILE *pimout_read;
// Open file in SUT mode:
pimout_read = fopen("output.sut","rb");
 //generate size pointer <Dynamic >
fread(imout,sizeof(char),bit_end,pimout_read);
 fclose(pimout_read);
////////////////////////////////////////////////////////////////////
// Set r_x,r_y (size Data input)
////////////////////////////////////////////////////////////////////
long int count_r = 0; int i,r_i,r_j;
//generate size pointer <Dynamic >
 float **fdwt = new float*[s_x];
 for(  i=0; i<s_x; i++)
 {fdwt[i] = new float[s_y];}
 for (r_i = 0; r_i < s_y ; r_i++){
 for (r_j = 0; r_j < s_x ; r_j++)
 {fdwt[r_j][r_i] = (float)0; //data out -> double
 count_r = count_r + 1;}}
 ////////////////////////////////////////////////////////////////////
 /* SPIHT -> Decode
 ////////////////////////////////////////////////////////////////////*/
 //Decode_SPIHT + LFC
SPIHT_Decode(LFC_list_type,fdwt,imout,bit_end,s_x,s_y,l
evel,n);
////////////////////////////////////////////////////////////////////
//// IDWT
bi_idwt2(fdwt,r_xx/16,r_yy/16);
bi_idwt2(fdwt,r_xx/8,r_yy/8);
bi_idwt2(fdwt,r_xx/4,r_yy/4);
bi_idwt2(fdwt,r_xx/2,r_yy/2);
bi_idwt2(fdwt,r_xx,r_yy);
printf("\nbit rate = %0.5f bpp -> bit compress = %d bit\n",
(float)bit_end/(s_x*s_y),bit_end);
round_image_abs(fdwt,r_xx,r_yy);
////////////////////////////////////////////////////////////////////
// Transfer Data from VC++        to MATLAB // Set
w_x,w_y in constant.h (size Data input)
////////////////////////////////////////////////////////////////////
FILE *p_write;
long int  numwritten;
long int count_w = 0;
 //generate size pointer <Dynamic >
unsigned char *w_list = new unsigned char[r_xx*r_yy]; //
Open file in text mode:
p_write = fopen( "out_image.dat", "wb" );
for ( int w_i = 0; w_i < s_y; w_i++ ){
for (int w_j = 0; w_j < s_x ; w_j++){
w_list[count_w] = (unsigned char)(fdwt[w_j][w_i]);
count_w = count_w + 1; } }
numwritten = fwrite(w_list, sizeof(char),s_x*s_y, p_write );
fclose( p_write );
////////////////////////////////////////////////////////////////////
//clear Memory pointer
delete[] w_list;
return(0);}
```

โปรแกรม Decode_SPIHT_main.cpp

```
/*-------------------------------------------------------*/
#include"SPIHT.h"
void SPIHT_Decode(char* LFC_list_type,float**
output_image,char* imout,int bit_end,int s_x,int s_y,int
level,int n){
//Initial DATA i_SPIHT = output_image (size = s_x*s_y)
```

```
int i,j,oi,oj;
for (i=0;i<s_x;i++){for(j=0;j<s_y;j++)
{ output_image[i][j] = 0; } }
//%1)Initialization :output n = [log2(max(i,j){|ci,j|})]
int nl,first_x,first_y; float num_n;
double number;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
nl = (int)number;
first_x = s_x/(nl); first_y = s_y/(nl);
number = pow((double)2,(double)n);
num_n = (float)number;
long count,count_LSP,count_LIP,count_LIS;
count = 0; count_LSP = 0;
count_LIP = 0; count_LIS = 0;
//%Initial LSP size=[s_x*s_y 3]
int* LSP_x = new int[long(s_x*s_y/2)];
int* LSP_y = new int[long(s_x*s_y/2)];
int* LSP_abs = new int[long(s_x*s_y/2)];
//%Initial LIP size=[s_x*s_y 2]
int* LIP_x = new int[s_x*s_y/2];
int* LIP_y = new int[s_x*s_y/2];
for (i=0 ; i < first_x ; i++)
{for (j=0 ; j < first_y ; j++)
{ LIP_x[count_LIP] = i;
LIP_y[count_LIP] = j;
count_LIP = count_LIP + 1; } }
//%Initial LIS size=[s_x*s_y 3]
int* LIS_x = new int[s_x*s_y/2];
int* LIS_y = new int[s_x*s_y/2];
int* LIS_type = new int[s_x*s_y];
for ( i=0 ; i < first_x ; i++)
{ for (j=0 ; j < first_y ; j++){
//Check odd and even
oi = i%2;  oj = j%2;
//Check i
if (oi == 0){ oi = 0; } else{ oi = 1; }
//Check j
if (oj == 0){ oj = 0; } else{ oj = 1; }
if ( (oi+oj)!= 0 ) {
LIS_x[count_LIS] = i; LIS_y[count_LIS] = j;
LIS_type[count_LIS] = 0; //%->set type A
count_LIS = count_LIS +1; } }  }
//%1.2 Make it LFC list , size of LFC list
int count_LFC; count_LFC = (level*4*3) + 4;
int* LFC_type = new int[count_LFC];
for(int e=0; e <count_LFC ;e++)
{ LFC_type[e] = (int)LFC_list_type[e]; }
for (int step_Quantization = 1; step_Quantization <=
step_quan ; step_Quantization++)  {
//%2)Sorting pass:
//%2.1 for each entry (i,j) in the LIP
for (int t=0; t < count_LIP; t++)
//%2.1.1 output Sn(i,j);
{if (bit_end <= count){break;}
int output; output = imout[count]; count = count + 1;
//%2.1.2)  Sn(i,j) = 1 then move (i,j) to the LSP and output
the sign of cij
if (output == 1) {
LSP_x[count_LSP]  = LIP_x[t];
LSP_y[count_LSP]  = LIP_y[t];
LSP_abs[count_LSP] = (int)num_n;
//%Output the sign of cij
output = imout[count];
count = count + 1;
if (output == -1) {
LSP_abs[count_LSP] = -LSP_abs[count_LSP];}
//%END of Output the sign of cij
count_LSP = count_LSP+1;
LIP_x[t] = -1;
LIP_y[t] = -1;} //%End of the Sn(i,j)=1
if (bit_end <= count){break} //%END of is Sn(i,j) = then
move (i,j) to the LSP
//%2.1.2)  Move LIP(i,j) at sn(i,j)=1
long* count_LIP_new = new long[1];
cut_LIP(count_LIP_new,LIP_x,LIP_y,count_LIP);
count_LIP = count_LIP_new[0];
//% END of Loop for each entry (i,j) in the LIP
//%2.2 for each entry (i,j) in the LIS
//%Initial parameter 2.2
long count_LIS_old;
count_LIS_old = count_LIS;
int change,set_add;
```

```
change = 100;  set_add = 0;
for (long LIS_length = 0 ; LIS_length < 100000000 ;
LIS_length ++){
if ((change == 0) && (count_LIS_old <= LIS_length))
{break;}
if (LIS_length==0){change = 0;}
if ((count_LIS_old <= LIS_length)&& set_add==1){change
= change - 1;}
//%Find Do(i,j) , LO(i,j)
i = LIS_x[LIS_length];  j = LIS_y[LIS_length];
//%Check LFC list
int LFC_D = 0; //%LFC O(i,j)
int LFC_L = 0; //%LFC L(i,j)
//i_coor_descendants_gen
int value_SD,value_SL,Lij_set;
int *type_ij = new int[1];//O_list
int **DOij_list = new int*[4];
for(int  ii=0; ii<4; ii++){ DOij_list[ii] = new int[2]; }
int LFC_x,LFC_y,Let_ijk=0;
coor_offspring(DOij_list,i,j,s_x,s_y,level);
LFC_x = DOij_list[0][0]; LFC_y = DOij_list[0][1];
int type_LFC_label;
type_LFC_label = type_label(LFC_x,LFC_y,s_x,
s_y,level);
if (LFC_type[type_LFC_label] <= step_Quantization)
{LFC_D = 1; //%->No LFC D(i,j)}
int **O_listo = new int*[4];
for(ii=0; ii<4; ii++)
{O_listo[ii] = new int[2];}
coor_offspring(O_listo,LFC_x,LFC_y,s_x,s_y,level);
int LFC_ox,LFC_oy;
if (O_listo[0][0] != -1){Let_ijk = 1;
LFC_ox = O_listo[0][0]; LFC_oy = O_listo[0][1];
type_LFC_label = type_label(LFC_ox,LFC_oy,s_x,
s_y,level);
if (LFC_type[type_LFC_label] <= step_Quantization)
{ LFC_L = 1; //%->No LFC L(i,j) }}
int check_work = 0; int* count_set = new int[1];
if ((LFC_D + LFC_L) != 0){
i_coor_descendants_s(count_set,DOij_list,type_ij,i,j,s_x,s_y
,level);
Lij_set  = type_ij[0];
check_work = 1; }
int absx,d_fdwt,yp;
int check_LIP = 0;
if (check_work == 1){
//%2.2.1 if the entry is of type A
if (LIS_type[LIS_length] == 0)//%Output Sn(D(i,j));
{if (bit_end <= count){goto a;}
value_SD = imout[count];
count = count + 1;
if (value_SD == 1){
for (int OD_A=0 ; OD_A < count_set[0] ; OD_A++)
//%Output Sn(OD(k,l)){absx = imout[count];
count = count + 1;
if (absx == 1)//%Sn(O(i,j)) is significant
{LSP_x[count_LSP] = DOij_list[OD_A][0];
LSP_y[count_LSP] = DOij_list[OD_A][1];
LSP_abs[count_LSP] = (int)num_n;
//%Output sing of c(k,l)
d_fdwt = imout[count]; count = count + 1;
if (d_fdwt == -1)
{LSP_abs[count_LSP] = -LSP_abs[count_LSP];}
count_LSP = count_LSP+1;
if (bit_end <= count){goto a;}}
else //%Sn(O(i,j)) is insignificant
{LIP_x[count_LIP] = DOij_list[OD_A][0];
LIP_y[count_LIP] = DOij_list[OD_A][1];
count_LIP = count_LIP + 1;  check_LIP = check_LIP + 1;
if (bit_end <= count){goto a;} }}
if (Let_ijk == 1)//%remove (i,j) from the LIS (type A)
{if (check_LIP == count_set[0]){
//%set type A
for(yp = 0 ; yp < count_set[0] ; yp++){
LIS_x[count_LIS]  = DOij_list[yp][0];
LIS_y[count_LIS]  = DOij_list[yp][1];
LIS_type[count_LIS] = 0; count_LIS = count_LIS + 1;
change = change + 1;}
set_add = 1;
//%remove (i,j) from the LIS_B
LIS_x[LIS_length]  = -1;  LIS_y[LIS_length]  = -1;
LIS_type[LIS_length] = 0; }
else{ if (LFC_L == 1) //%Output Sn(L(i,j))
{if (bit_end <= count){goto a;}
```

```
value_SL  = imout[count]; count = count + 1;
if (value_SL == 1){ //%set type A
for(int yp = 0 ; yp < count_set[0] ; yp++){
LIS_x[count_LIS]  = DOij_list[yp][0];
LIS_y[count_LIS]  = DOij_list[yp][1];
LIS_type[count_LIS] = 0;
count_LIS = count_LIS + 1;
change = change + 1; }
//%remove (i,j) from the LIS_B
LIS_x[LIS_length]  = -1;
LIS_y[LIS_length]  = -1;
LIS_type[LIS_length] = 1;
set_add = 1; }
else{LIS_type[LIS_length] = 1;}
if (bit_end <= count){goto a;} }
else{LIS_type[LIS_length] = 1; } } }
else{//%remove (i,j) from the LIS_B
LIS_x[LIS_length] = -1;  LIS_y[LIS_length] = -1;
LIS_type[LIS_length] = 1;  } }
}//%END of entry is of type A
else { if (LFC_L == 1) { //%Output Sn(L(i,j))
if (bit_end <= count){goto a;}
value_SL  = imout[count]; count = count + 1;
if (value_SL == 1) {
for(yp = 0 ; yp < count_set[0] ; yp++) {
LIS_x[count_LIS] = DOij_list[yp][0];
LIS_y[count_LIS]  = DOij_list[yp][1];
LIS_type[count_LIS] = 0;
count_LIS = count_LIS + 1;
change = change + 1; }
set_add = 1;
//%remove (i,j) from the LIS_B
LIS_x[LIS_length]  = -1;
LIS_y[LIS_length]  = -1;
LIS_type[LIS_length] = 0;
if (bit_end <= count){goto a;} }
} } } }//END of LOOP LIS
long* count_LIS_new = new long[1];
cut_LIS(count_LIS_new,LIS_x,LIS_y,LIS_type,count_LIS)
;
count_LIS = count_LIS_new[0];
//%3 Refinement Pass: for each entry (i,j) in the LSP
long* count_new = new long[1];
i_refinement_pass(count_new,imout,LSP_abs,count_LSP,
(num_n/2),bit_end,count);
count = count_new[0];
//%4 Ouantization
num_n = num_n/2;}
a:
re_image_pass(output_image,LSP_x,LSP_y,LSP_abs,count
_LSP);
delete[] LSP_x;  delete[] LSP_y;
delete[] LSP_abs; delete[] LIP_x;
delete[] LIP_y;  delete[] LIS_x;
delete[] LIS_y; delete[] LIS_type; }
```

โปรแกรม Decode_SPIHT_function.cpp

```
/*-------------------------------------------------------*/
#include"SPIHT.h"
////////////////////////////////////////////////
//SPIHT-Decode-function
////////////////////////////////////////////////
void i_coor_descendants_s(int *count_set,int** O_list,int*
type_ij,int i,int j,int s_x,int s_y,int level)
{/*Input <i,j = coordinates>
%Output <Lij_set  = set of L(i,j)>
%          = 0 -> No membership
%          = 1 -> Yes membership
%type_ij -> type_ij[0] = Lij_set; */
//Initial parameter
int Lij_set  = 1;  int null = 1;
%limit of first Layer and finit Layer
%first Layer */
double number;
int  p,first_x,first_y,finit_x,finit_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p);  first_y = s_y/(p);
//%finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//%Group 1 ->DWT Level 1 ,LH1 ,HL1 ,HH1
```

```
if ((i>= finit_x/2 && i<s_x && j>=0 && j<s_y) || (i>=0
&& i<finit_x/2 && j>=finit_y/2 && j<s_y))
{ Lij_set = 0;} //O_list
coor_offspring_s(count_set,O_list,i,j,s_x,s_y,level);
type_ij[0] = Lij_set; }
////////////////////////////////////////////////
void i_coor_descendants(int** O_list,int* type_ij,int i,int
j,int s_x,int s_y,int level)
{ /*Input
%i,j = coordinates
%Output
%Lij_set  = set of L(i,j)
%            = 0 -> No membership
%            = 1 -> Yes membership
%type_ij
type_ij[0] = Lij_set; */
//Initial parameter
int Lij_set  = 1;
int null = 1;
%limit of first Layer and finit Layer
%first Layer */
double number; int  p,first_x,first_y,finit_x,finit_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number;
first_x = s_x/(p);  first_y = s_y/(p);
//%finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//%Group 1 ->DWT Level 1 ,LH1 ,HL1 ,HH1
if ((i>= finit_x && i<s_x && j>=0 && j<s_y) || (i>=0 && i
<finit_x && j>=finit_y && j<s_y))
{Lij_set = 0; null = 0;}
if (null != 0){coor_offspring(O_list,i,j,s_x,s_y,level);}
type_ij[0] = Lij_set; }
////////////////////////////////////////////////
void i_refinement_pass(long* count_new,char* imout,int*
LSP_abs,long count_LSP,float num_n,long bit_end,long
count){int ans;
for (int t=0 ; t < count_LSP ; t++) {
if (bit_end <= count){break;}
ans = imout[count]; count = count + 1;
if (LSP_abs[t] >= 0) {
LSP_abs[t]  = (int)(LSP_abs[t] + num_n*ans);}
else{LSP_abs[t]  = (int)(LSP_abs[t] - num_n*ans);}
if (bit_end <= count){break;}}
count_new[0] = count;}
////////////////////////////////////////////////
void re_image(int** output_image,int* LSP_x,int*
LSP_y,int * LSP_abs,int* LSP_sign,long count_LSP)
{int i,j;
float s_amp;
for(long step = 0; step < count_LSP ; step ++) {
i = LSP_x[step];  j = LSP_y[step];
s_amp = LSP_abs[step] * (float)LSP_sign[step];
output_image[i][j] = (int)s_amp; }  }
////////////////////////////////////////////////
void re_image_pass(float** output_image,int* LSP_x,int*
LSP_y,int * LSP_abs,long count_LSP)
{ int i,j;
for(long step = 0; step < count_LSP ; step ++) {
i = LSP_x[step];  j = LSP_y[step];
output_image[i][j] = (float)LSP_abs[step]; } }
////////////////////////////////////////////////
void coor_offspring_s(int* countO,int** O_list,int i,int j,int
s_x,int s_y,int level)
{ i = i + 1;  j = j + 1; double number;  int count_O = 0;
int p,first_x,first_y,finit_x,finit_y,oi,oj;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number; first_x = s_x/(p);  first_y = s_y/(p);
//finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//Find odd and even output = 1 -> odd
                    //output = 0 -> even
//Check i
oi = i%2; oj = j%2;
if (oi == 0){oi = 0;} else{oi = 1;}
//Check j
if (oj == 0){oj = 0;} else{oj = 1;}
int Check = 1;
//No root if ->Yes L(i,j)
//Level END
```

```
int px,py;
 if (i>=1 && i<=first_x && j>=1 && j<=first_y) {
 if ((oi+oj)!=2) {
px=first_x-1;  py=first_y-1;
 if (oi==1 && oj==0)
{ O_list[0][0] = i; O_list[0][1] = j+py; }
 if (oi==0 && oj==1)
{ O_list[0][0] = i+px;  O_list[0][1] = j; }
 if (oi==0 && oj==0)
{ O_list[0][0] = i+px; O_list[0][1] = j+py; }
Check = 0; }
else { count_O = 0;
O_list[count_O][0] = i+1-1;
O_list[count_O][1] = j-1;
 //%1)
O_list[count_O+1][0] = i-1;
O_list[count_O+1][1] = j+1-1;
 //%2)
O_list[count_O+2][0] = i+1-1;
O_list[count_O+2][1] = j-1+1;
count_O = 3;
Check = 2; } }
//NO root if -> No L(i,j)
//Level 1 (LH1 HL1 HH1)
int a_1,a_2;
a_1 = (finit_x+1); a_2 = (finit_y+1);
if ((i>= a_1 && i<=s_x && j>=1 && j<=s_y) || (i>=1 && i
<=finit_x && j>= a_2 && j<=s_y) ) {
O_list[0][0] = -1;       //(-1)
O_list[0][1] = -1;       //(-1)
Check = 2 }
//Root of O(i,j) ->Yes L(i,j)
if (Check == 1) {
O_list[0][0]=(int)((i*2)-1); O_list[0][1]=(int)((j*2)-1);}
if (Check !=2){
O_list[0][0]-=1;  O_list[0][1]-=1;
//Find root of O(i,j)
//2 3 ;*=O(i,j)
//1)
O_list[1][0] = O_list[0][0]; O_list[1][1] = O_list[0][1] + 1;
//2)
O_list[2][0] = O_list[0][0] + 1; O_list[2][1] = O_list[0][1];
//3)
O_list[3][0] = O_list[0][0] + 1;
O_list[3][1] = O_list[0][1] + 1;
count_O = 4; }
countO[0] = count_O; }
////////////////////////////////////////////////
void coor_offspring(int** O_list,int i,int j,int s_x,int s_y,int
level)
{ i = i + 1;  j = j + 1;
double number; int  p,first_x,first_y,finit_x,finit_y;
//limit of first Layer and finit Layer
//first Layer
number = pow((double)2,(double)level);
p = (int)number; first_x = s_x/(p);  first_y = s_y/(p);
//finit Layer
finit_x = s_x/2; finit_y = s_y/2;
//Find odd and even
//output = 1 -> odd output = 0 -> even
 int oi,oj;
 //Check i
 oi = i%2; oj = j%2;
 if (oi == 0){oi = 0;} else{oi = 1;}
 //Check j
if (oj == 0){oj = 0;} else{oj = 1;}
int Check = 1;
//No root if ->Yes L(i,j)
//Level END
int px,py;
if (i>=1 && i<=first_x && j>=1 && j<=first_y)
{ if ((oi+oj)!=2) {
px=first_x-1;  py=first_y-1;
if (oi==1 && oj==0)
{O_list[0][0] = i;  O_list[0][1] = j+py; }
if (oi==0 && oj==1)
{O_list[0][0] = i+px;  O_list[0][1] = j; }
if (oi==0 && oj==0)
{ O_list[0][0] = i+px; O_list[0][1] = j+py; }
Check = 0; }
else { O_list[0][0] = -1; O_list[0][1] = -1
Check = 2; } }
 //NO root if -> No L(i,j)
 //Level 1 (LH1 HL1 HH1)
```

```
int a_1,a_2;
a_1 = (finit_x+1); a_2 = (finit_y+1);
if ((i>= a_1 && i<=s_x && j>=1 && j<=s_y) || (i>=1 && i
<=finit_x && j>= a_2 && j<=s_y) )
{ O_list[0][0] = -1;    //(-1)
O_list[0][1] = -1;      //(-1)
Check = 2; }
//Root of O(i,j) ->Yes L(i,j)
if (Check == 1)
{ O_list[0][0]=(int)((i*2)-1);
O_list[0][1]=(int)((j*2)-1); }
if (Check !=2) {
O_list[0][0]-=1;  O_list[0][1]-=1;
//Find root of O(i,j)
//2 3 ;*=O(i,j)
//1)
O_list[1][0] = O_list[0][0];  O_list[1][1] = O_list[0][1] + 1;
//2)
O_list[2][0] = O_list[0][0] + 1; O_list[2][1] = O_list[0][1];
//3)
O_list[3][0] = O_list[0][0] + 1;
O_list[3][1] = O_list[0][1] + 1; }}
////////////////////////////////////////////////
void cut_LIS(long* count_LIS_new,int* LIS_x,int*
LIS_y,int* LIS_type,long count_LIS)
{ long count=0;
for(long z=0 ; z < count_LIS ; z++)
{ if(LIS_x[z]!=-1)
{ count = count + 1; } }
int* x_new = new int[count];
int* y_new = new int[count];
int* z_new = new int[count];
count = 0;
for( z=0 ; z < count_LIS ; z++) {
if(LIS_x[z]!=-1) {
x_new[count] = LIS_x[z];
y_new[count] = LIS_y[z];
z_new[count] = LIS_type[z];
LIS_x[z] = -1;
count = count + 1; } }
for( z=0 ; z < count ; z++)
{LIS_x[z]    = x_new[z];
LIS_y[z] = y_new[z];
LIS_type[z]  = z_new[z]; }
count_LIS_new[0] = count;
delete[] x_new;  delete[] y_new; delete[] z_new;}
////////////////////////////////////////////////
void cut_LIP(long* count_LIP_new,int* LIP_x,int*
LIP_y,long count_LIP)
{ long count=0;
for(long z=0 ; z < count_LIP ; z++) {
if(LIP_x[z]!=-1) { count = count + 1; } }
int* x_new = new int[count]; int* y_new = new int[count];
count = 0;
for( z=0 ; z < count_LIP ; z++)
{ if(LIP_x[z]!=-1) {
x_new[count] = LIP_x[z]; y_new[count] = LIP_y[z];
LIP_x[z]    = -1;
count = count + 1; } }
for( z=0 ; z < count ; z++) {
LIP_x[z] = x_new[z]; LIP_y[z] = y_new[z]; }
count_LIP_new[0] = count;
delete[] x_new; delete[] y_new; }
////////////////////////////////////////////////
void round_image_abs(float** input,int s_x,int s_y)
{ float point=0.5;
int store_int;
for(int i = 0 ; i <s_x  ;i++)
{ for(int j = 0 ; j<s_y ;j++) {
 if (input[i][j] > 0)  { if (input[i][j] <= 255)
 { store_int = (int)(input[i][j] + point);
 input[i][j] = (float)store_int; }
 else {input[i][j] = (float)(255);}}
if (input[i][j] <= 0)
{input[i][j] = 0; } } } }
```

โปรแกรม bi_dwt.cpp

เหมือนภาคผนวก ก.

โปรแกรม LFC.cpp

เหมือนในภาคผนวก ก.

ประวัติผู้วิจัย

กิตติ อัตถกิจมงคล เกิดเมื่อ วันที่ 7 สิงหาคม พ.ศ. 2515 ที่ จ. สตูล ปัจจุบันดำรงค์ตำแหน่งผู้ช่วยศาสตราจารย์ และเป็นอาจารย์ประจำสาขาวิชาวิศวกรรมไฟฟ้า สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จบการศึกษาระดับปริญญาตรี วศ.บ. วิศวกรรมอีเล็กทรอนิกส์ จากสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง พ.ศ. 2537 ระดับปริญญาโท M.S. Electrical Engineering และ ปริญญาเอก Ph.D. Electrical Engineering จาก Vanderbilt University สหรัฐอเมริกา เมื่อ พ.ศ. 2539 และ พ.ศ. 2542 ตามลำดับ งานวิจัยที่สนใจคือ การประมวลผลสัญญาณดิจิตอล การประมวลผลสัญญาณภาพ การแปลงเวฟเล็ต การแปลงมัลติเวฟเล็ตและการประยุกต์ใช้ สถานที่ติดต่อ สาขาวิชาวิศวกรรมไฟฟ้า สำนักวิชาวิศวกรรมศาสตร์ มหาวิยาลัยเทคโนโลยีสุรนารี อ. เมือง จ. นครราชสีมา 30000 หรือ kitti@ccs.sut.ac.th

ณัฐนันท์ ทัดพิทักษ์กุล เกิดเมื่อ วันที่ 20 พฤศจิกายน พ.ศ. 2521 ที่ กรุงเทพมหานคร ปัจจุบันดำรงค์ตำแหน่งผู้ช่วยวิจัย ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ จบการศึกษาระดับปริญญาตรี วศ.บ. วิศวกรรมโทรคมนาคม ระดับปริญญาโท วิศวกรรมไฟฟ้า จาก มหาวิยาลัยเทคโนโลยีสุรนารี เมื่อ พ.ศ. 2543 และ พ.ศ. 2545 ตามลำดับ งานวิจัยที่สนใจคือ การประมวลผลสัญญาณดิจิตอล การประมวลผลสัญญาณภาพ การแปลงเวฟเล็ตและการประยุกต์ใช้ สถานที่ติดต่อ ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ 112 ถ.พหลโยธิน ต. คลองหนึ่ง อ. คลองหลวง จ. ปทุมธานี 12120 หรือ Nattanun.Thatphithakkul@nectec.or.th