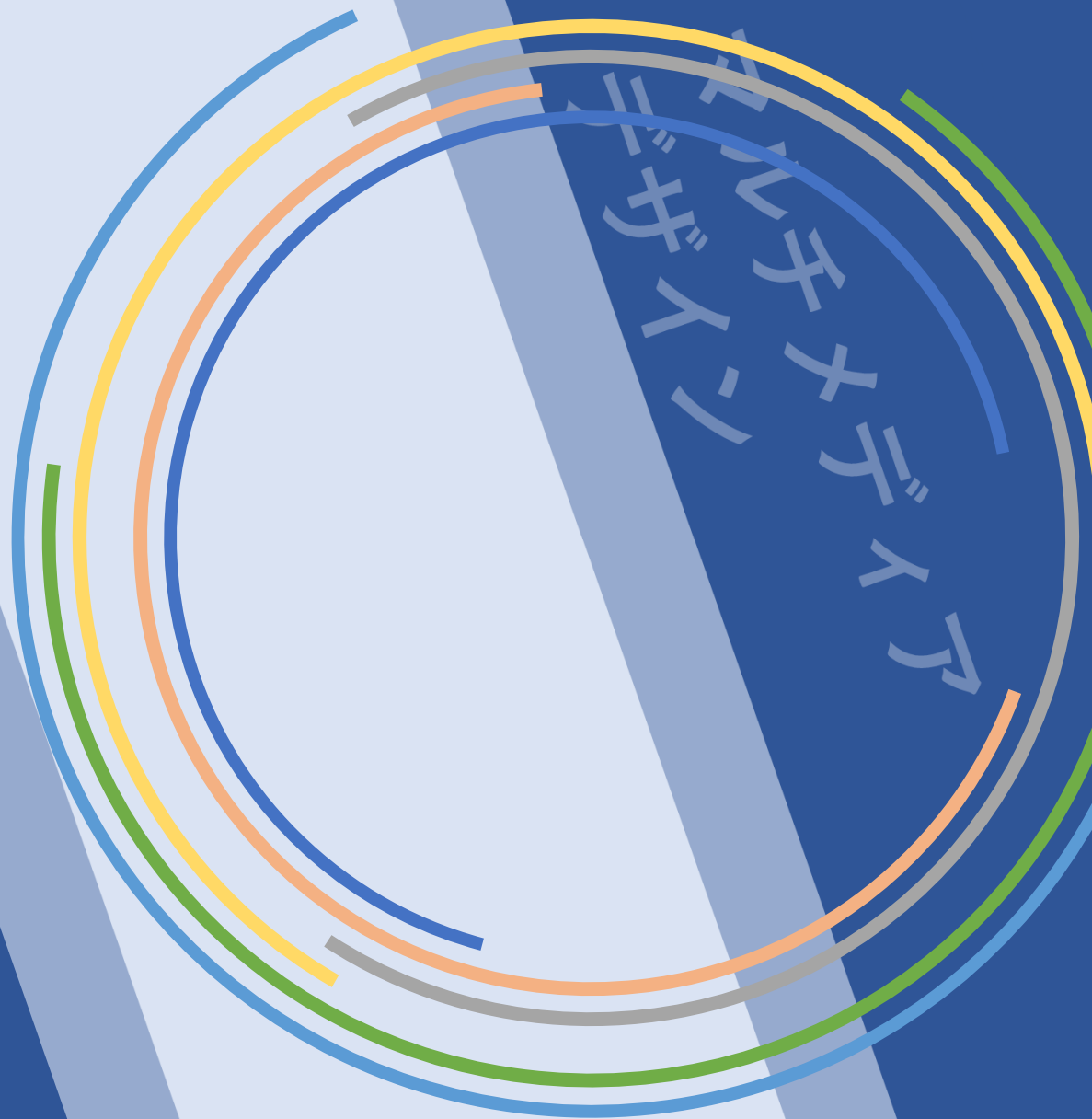


この授業のスライドはLMSで公開されます
実験の質問やレポートに関する質問などはSlackへ
<https://keio-st-multimedia.slack.com>



#9 圧縮・不可逆変換

担当： 西 宏章





ハーフトーニング

60

- 一般に自然画は連続諧調となり、これをうまく表現するために考案された最も単純な方法
- オリジナル画像を元に様々なハーフトーニング手法を比較する
- 単純2値化手法、ちょうど中間(128)を閾値として2値化しているが、全体のメディアン値で2値化してもよい



16	221	...
182	97	
⋮		

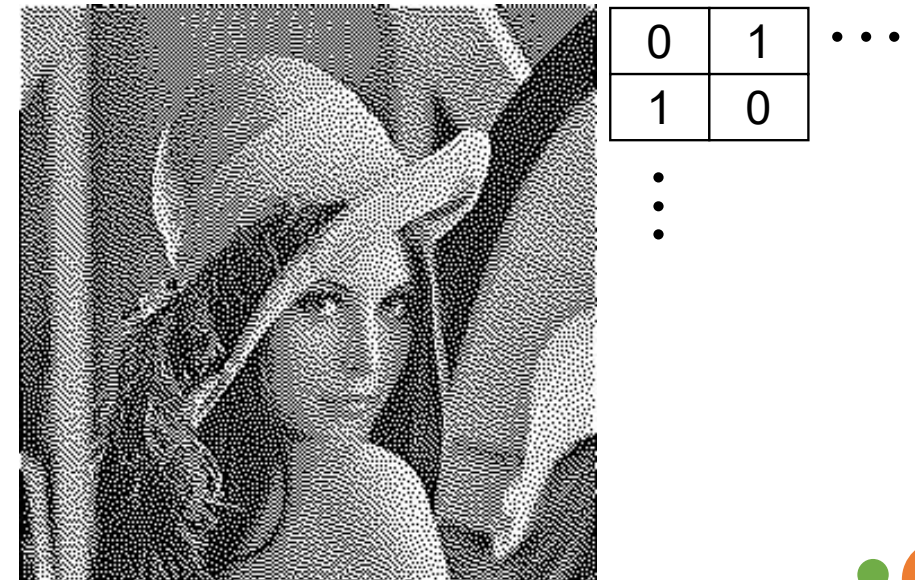


0	1	...
1	0	
⋮		



誤差拡散法

- 2値化の際の誤差を近隣ピクセルに拡散
- 拡散の仕方によりいろいろな手法がある。以下は一例
 - 最初は16なので、これを0としたため、221に16を足す
 - 次に221+16を1(=255)としたため、255-221-16を次の画素値から引く
- 不自然な文様が発生する可能性もある



組織的ディザ(Dither)法

- ディザ行列と原画像のタイルとの積を求めて画素を計算
 - ある画素を計算するために周辺の画素の影響を計算する
 - Jarvis, Judice & Ninke フィルタ
 - Floyd Steinberg フィルタ、
 - ブルーノイズやブレーノイズフィルタなど

現画像

ブルーノイズ
マスクHPFディザ
行列法グリーンノイズ
マスクR(M)PF

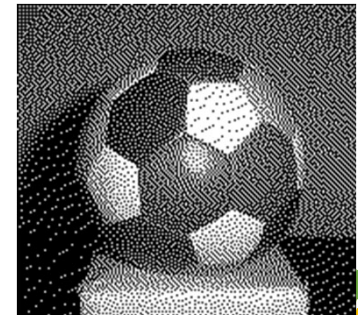
Jarvis, Judice & Ninke フィルタ

		注目	7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48



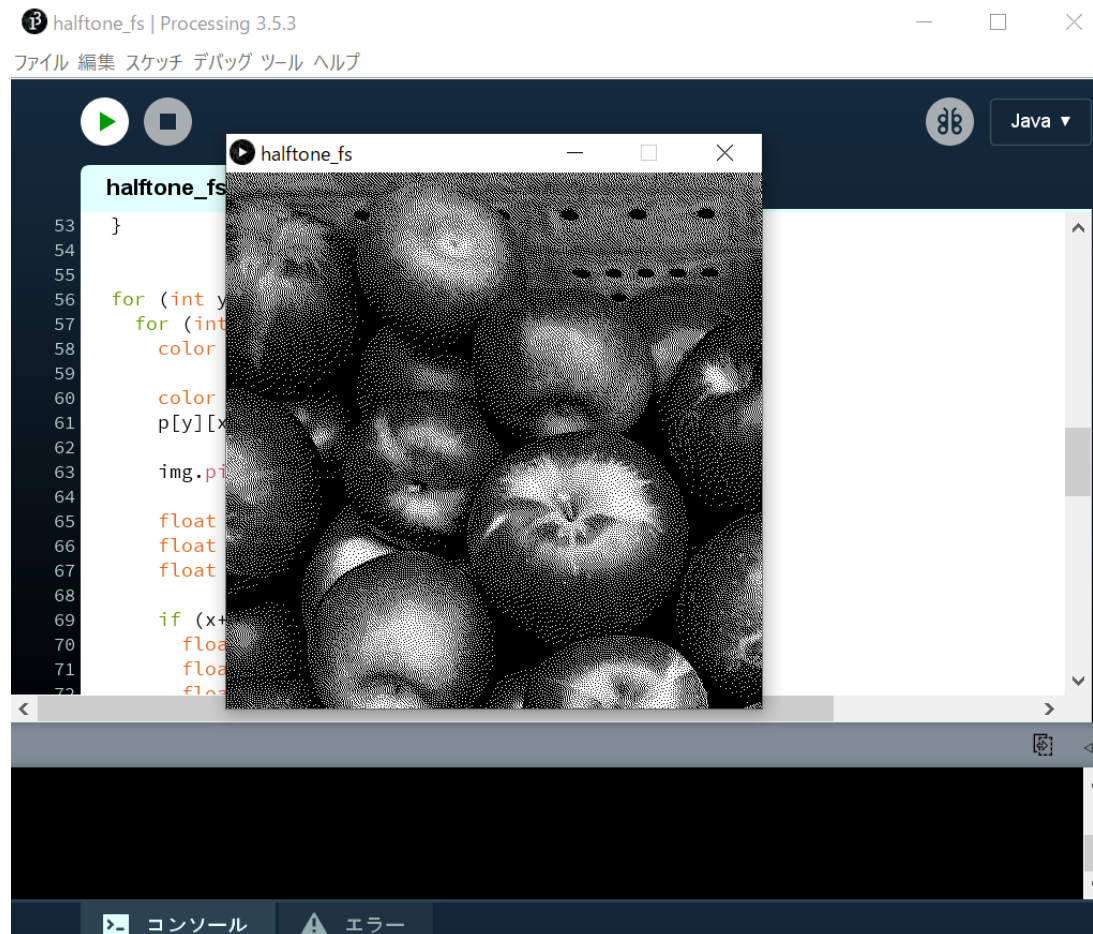
Floyd Steinberg フィルタ

	注目	7/16
3/16	5/16	1/16



Processing: Floyd-Steinberg Dither

63



クロマキー合成

64

- 特定の色成分を削除し、別画像と合成する手法
- 一般的に緑や青などの色が使われるが何でもよい
 - 服装や肌の色などに現れない色にする
 - ブルーバック合成では青






演習問題（7）

65

- Processingを用い誤差拡散法によるハーフトニング画像を作成しなさい
 - 好きなイメージ（300x300から500x500）の画像を準備する
 - 300x300の場合はsize(600,300)とし変換前と変換後の画像を表示できるようにする
 - 利用する画像の大きさに合わせて適宜sizeを調整すること
 - PImage img1, img2としてimg1(変換前)とimg2(変換後)を準備する
 - img1へloadImageを用いて画像を読み込む
 - このimg1に対して、filterを用いてグレースケールの画像にする
 - フィルタを使って一発で変換
 - グレースケール化画素値をもとに誤差拡散法による計算を行いimg2を作る
 - img2をまっさらで準備するため事前にcreateImageしておきなさい
 - サイズは準備した画像に合わせ、RGBを指定する
 - img1とimg2を2つ張り付けなさい
 - image(img1, 0, 0); image(img2, 300, 0);などとする





注意事項

- これらの設問に対する回答を、Microsoft Wordファイルで作成
- LMSで提出すること
- A4で作成すること
 - なお、常識的な範囲でページ数を超過することは問題ありません
- **ソースコード、動作画面をキャプチャして貼り付けること**
 - キャプチャはキャプチャしたいWindowを選択してAlt+Print Screenでできます。そのまま、Wordファイルに張り付けることができます（+は押しながらの意味）
 - MacOSは、Control+Command+Shift+4とするとカーソルがカメラ型に変わります。特定のウインドウやメニューバーをクリックしてキャプチャ、画像はクリップボードに転送されます
- 最初にタイトルとして「演習問題（7）」と書き、名前と学籍番号を記載すること
このフォーマットに従っていないレポート答案は受け取らない
- 締め切りなど詳細はLMSを確認すること





演習問題のヒント

67

- 実行結果の例を示します
- 前回と注意すべきところは同じです
 - 前回の課題や、今回配布したプログラムを元に作り替えるとよいです！
 - **一から作る必要はありません！！**
 - ググってもよい！
- フルカラー画像ですのでRGBが存在します
 - RGB同じ値が入っている画像になります





演習問題（8）

68

- Processingを用いてクロマキー合成を行いなさい
 - 2つの画像を準備する
 - 一つの画像は一部を「なしかしらの色」にする
 - その画像の緑の部分を、別の画像の同じ位置の画素値に置き換えなさい
 - クロマキー効果が得られていることを確認しなさい
- 以上をラスタスキャンで置き換えれば基準点(7点)
- 工夫により加点する
 - 例えば色の判定対して何かしら工夫をする
 - 置き換え対象を複数にする



注意事項

- これらの設問に対する回答を、Microsoft Wordファイルで作成
- LMSで提出すること
- A4で作成すること
 - ソースコードの記載はコピペになるため不要で、どのような入力値を使ったかや、実行結果を示せば十分です。なお、常識的な範囲でページ数を超過することは問題ありません
- **ソースコード、動作画面をキャプチャして貼り付けること**
 - キャプチャはキャプチャしたいWindowを選択してAlt+Print Screenでできます。そのまま、Wordファイルに張り付けることができます（+は押しながらの意味）
 - MacOSは、Control+Command+Shift+4とするとカーソルがカメラ型に変わります。特定のウインドウやメニューバーをクリックしてキャプチャ、画像はクリップボードに転送されます
- 最初にタイトルとして「演習問題（８）」と書き、名前と学籍番号を記載すること
このフォーマットに従っていないレポート答案は受け取らない
- 締め切りなど詳細はLMSを確認すること

差分符号化(DPCM)

70

- 画像を差分で表現する

10	21	29	41	50	61
15	25	34	45	56	66
20	30	41	51	60	69
25	36	44	55	65	74
30	41	50	61	71	82

原画像

10	11	8	12	9	11
15	10	9	11	11	10
20	10	11	10	9	9
25	11	8	11	10	9
30	11	9	11	10	11

隣り合う2つの画素の差分
(微分値)

- 予測値による符号化
 - 近傍の画素値から符号対象となる画素値を予測
 - 予測値と実際の画素値との誤差を符号化することで、ダイナミックレンジを小さくする

$$\text{予測誤差 } E = x - (a + b - c)$$

	b	c	
	a	x	

10	21	29	41	50	61
15	-1	1	-1	2	-1
20	0	2	-1	-2	-1
25	1	-3	1	1	0
30	0	1	0	0	2

$(c - b) \cong (x - a)$ として予測する



離散コサイン変換

- DCTの基本パターン
 - DCT係数の意味
 - 周波数成分が水平と垂直の2方向で表現
 - 基本パターンを含む度合いを表現

200	180	160	140
175	165	135	130
130	125	115	120
100	110	105	100

画素値

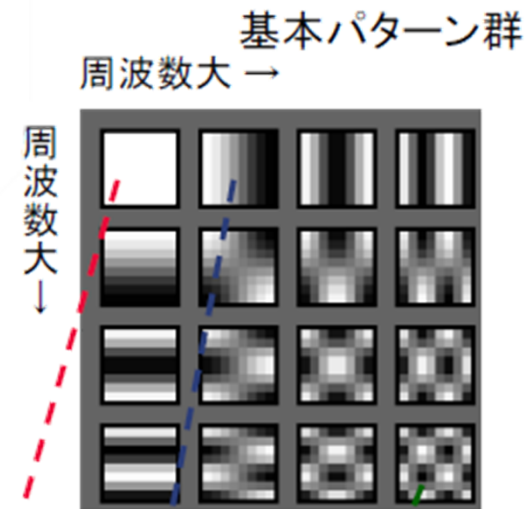
DCT

逆DCT

548	46	0	-6
102	36	4	3
0	0	-7	6
-2	-7	4	4

DCT係数値

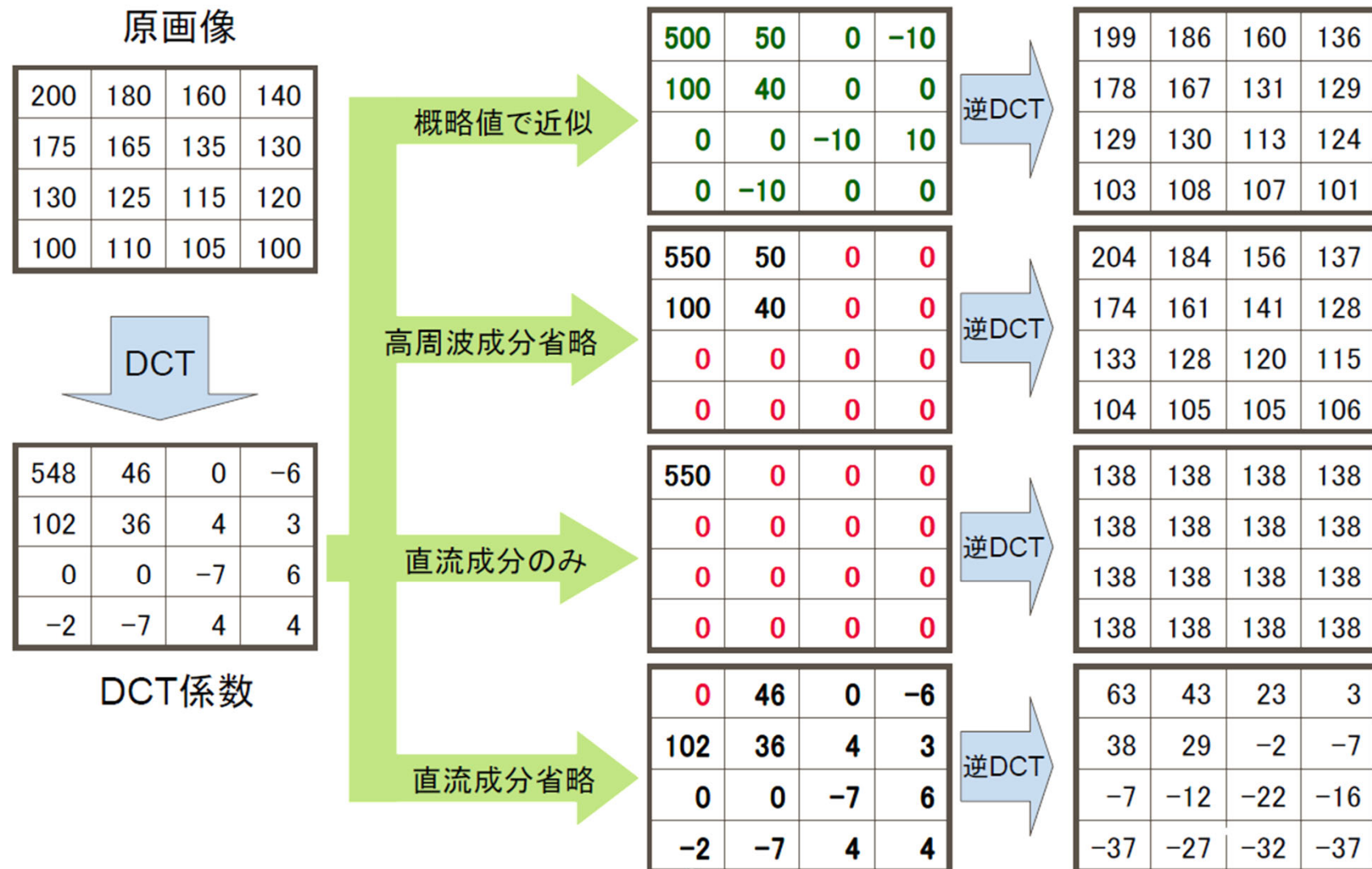
直流成分





離散コサイン変換

72



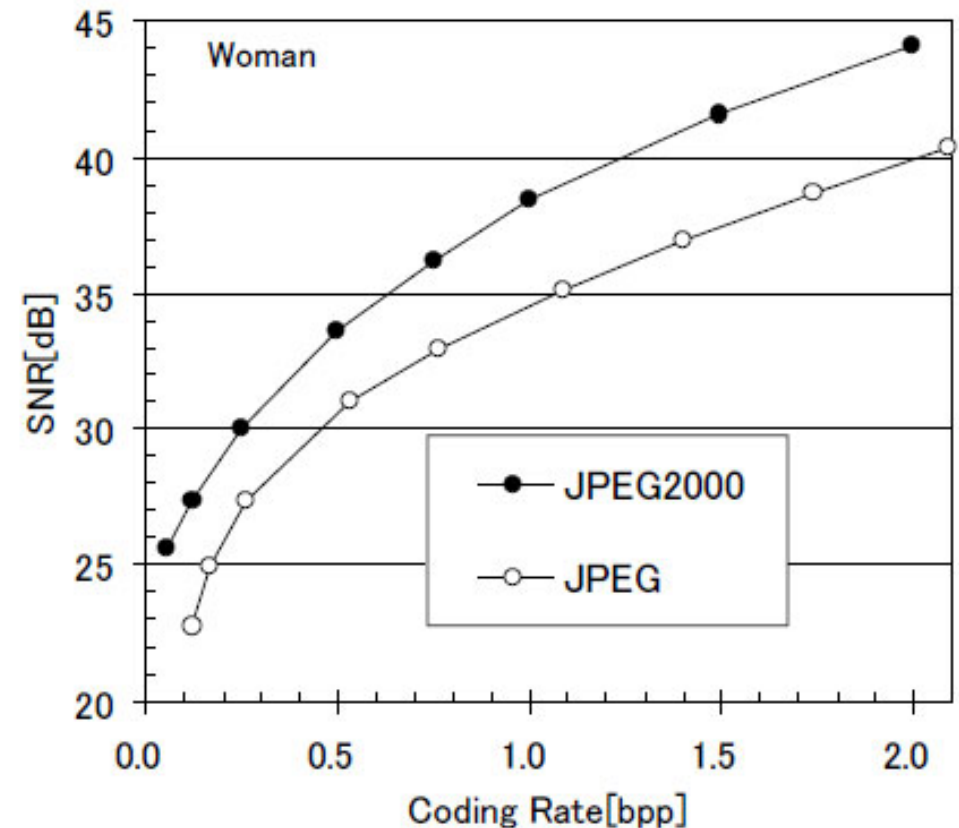
- JPEGアルゴリズムの特色
 - 人間の目の特性に着目して圧縮
 - 輝度変化に比べて色変化を認知する能力が低いことを利用し色差情報を間引く
 - 緩やかに変化する箇所の輝度／色の階調差には敏感だが、細かく変化する箇所では少ない階調でも不自然に感じないことを利用し、空間周波数分析における高周波数領域のデータを間引く
- JPEGの欠点
 - 高圧縮時にブロック状のひずみが目立つ
 - 圧縮時に画像を8*8のマスキに分け、それぞれに離散コサイン変換（DCT）をかけるため
 - マスキのつながりめが不連続
 - 可逆圧縮と非可逆圧縮に別のアルゴリズムを用いることから一貫性がない



JPEG2000

74

- JPEG 離散コサイン変換
- JPEG2000 離散ウェーブレット変換
 - 高い圧縮率
 - ブロックノイズが発生しにくい
 - Embedded Coding
progressive伝送下で任意の符号化レートにおいて、最適な画質を提供する
 - デジタルカメラでメモリがフルになっても、撮影済み画像データの各最後尾をある割合でカットすれば、高い符号長対画質比を保持して追加撮影用のメモリの余裕を作り出すことができる



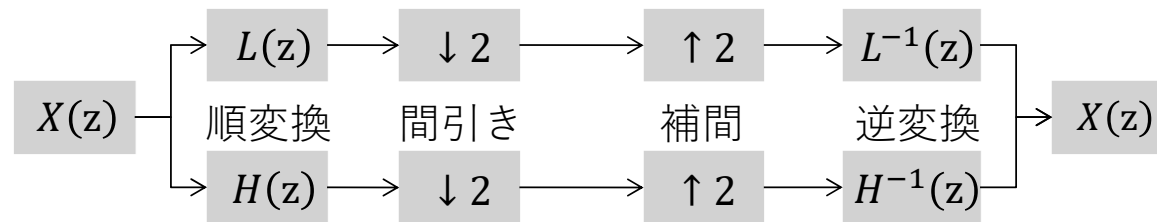
JPEG2000 vs JPEG

75



ウェーブレット変換

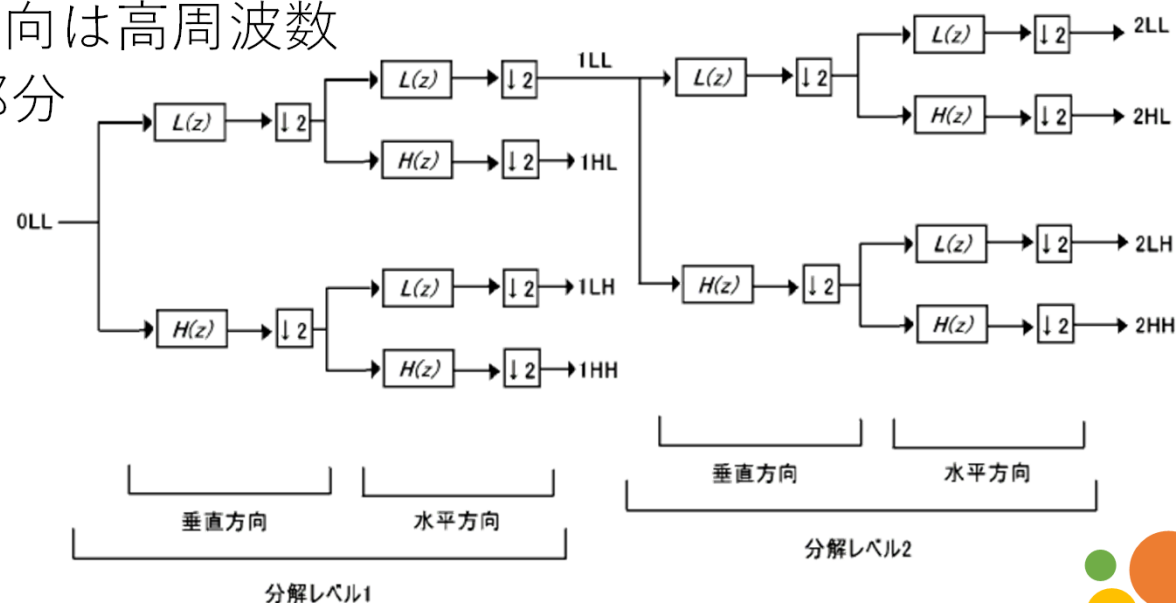
- マザーウェーブレットと呼ばれる波を、拡大縮小、並行移動して足し合わせることで与えられた入力波形を表現する手法
- 連続ウェーブレット変換は、（マザーウェーブレット g ）
 - $$W_g f(m, n) = \frac{1}{\sqrt{a^m}} \int_{-\infty}^{\infty} f(x) g\left(\frac{x - nb_0 a_0^m}{a_0^m}\right) dx$$
- 離散ウェーブレット変換は、
 - $$W_g f(m, n) = \frac{1}{\sqrt{a^m}} \sum_{k=-\infty}^{\infty} f(k) h\left(\frac{x - nb_0 a_0^m}{a_0^m}\right)$$
- 多重解像度解析
 - 周波数軸を等しい帯域幅で均等分割する
 - このフィルタを縦と横に用い2次元変換として扱う



2次元DWT

77

- 画像の垂直方向に対してハイパスおよびローパスフィルタをかけ、周波数の高い部分(H)と低い部分(L)に分ける
- H/Lそれぞれに対して、水平方向にハイパス・ローパスフィルタをかける
- 2次元DWT分解の操作から以下の結果が出力される。
 - LL … 水平、垂直ともに低周波数部分
 - HL … 水平方向は高周波数、垂直方向は低周波数
 - LH … 水平方向は低周波数、垂直方向は高周波数
 - HH … 水平、垂直ともに高周波数部分

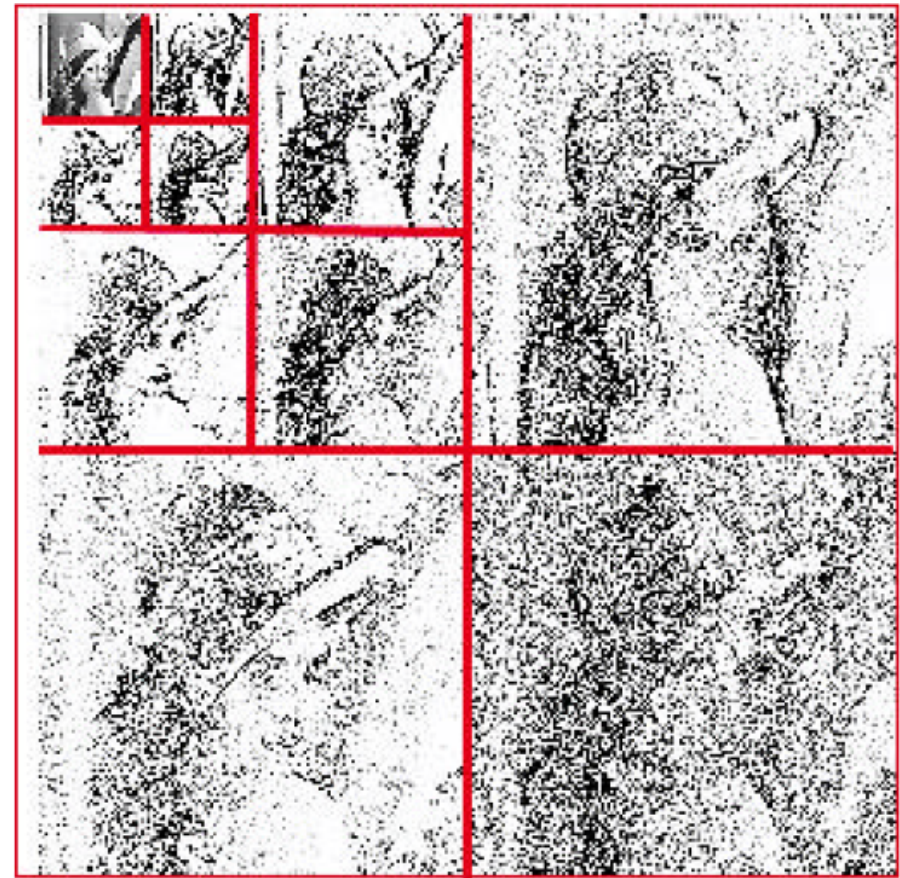
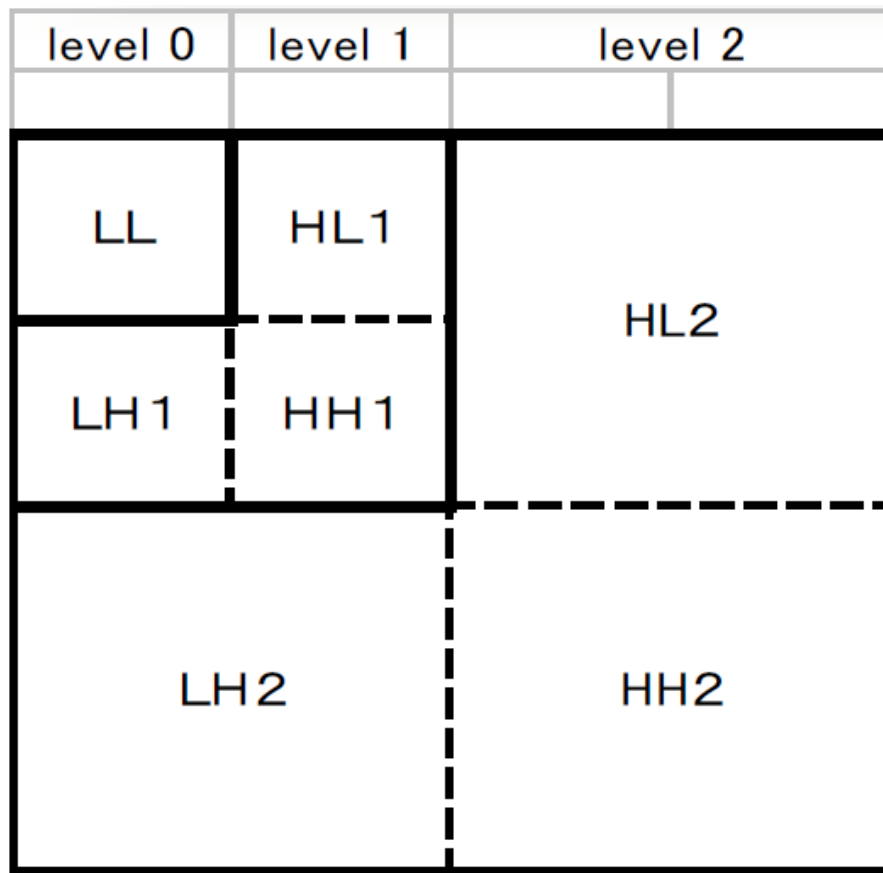




Wavelet変換

78

- $n \times n$ の領域でブロックを構成 (JPEGは 8×8)
- 比較的自由にブロックを構成できるためブロックノイズを抑制可能



- ROI:重要領域を先に送る工夫
 - 顔の部分を優先的に送るなど、注目する場所を優先的に展開



- Joint Photographic Experts Group(JPEG)が、JPEG2000の次にJPEG XRを発表、Windows Media Photoのルーツ
- 2018年にJPEGが、JPEG XSフォーマットを正式に発表
 - レイテンシの低下が特徴（ストリーミング向け、VRで違和感少ない）
 - 圧縮率は抑え気味、電力消費も少なく、JPEG2000は回路規模比較上100倍も必要
- AppleはJPEGからHEIFへと変更
- 動画はH.265/MPEG-H HEVC
 - ITUの規格、4K放送やUltra HD Blu-rayでも利用
 - H.264比で圧縮率が40%向上

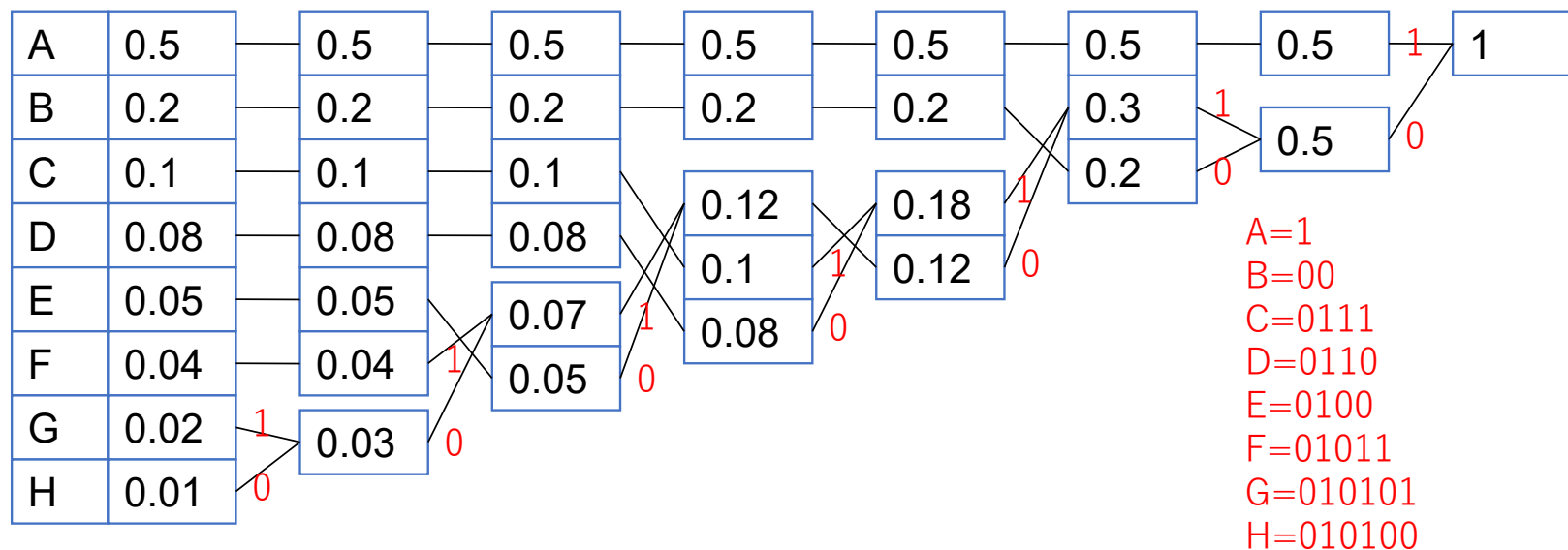




エントロピー符号化

80

- 一般的な情報圧縮技術を用いて圧縮（ハフマン符号）
- 情報の生起確率の大きいものから順に並べ、それぞれに符号を与えて圧縮する。原理は、「たくさん出現する情報程、短い情報量を利用すれば圧縮できる」
 - 画像圧縮では、たとえば、最も出現率の高い明度順などで圧縮





ハフマン符号の練習

81

- 次の発生確率の表をもとに全体の圧縮率を求めよ
- また、そのデータの部分列であるACBDAEABCAを圧縮し、その部分列における圧縮率を求めよ

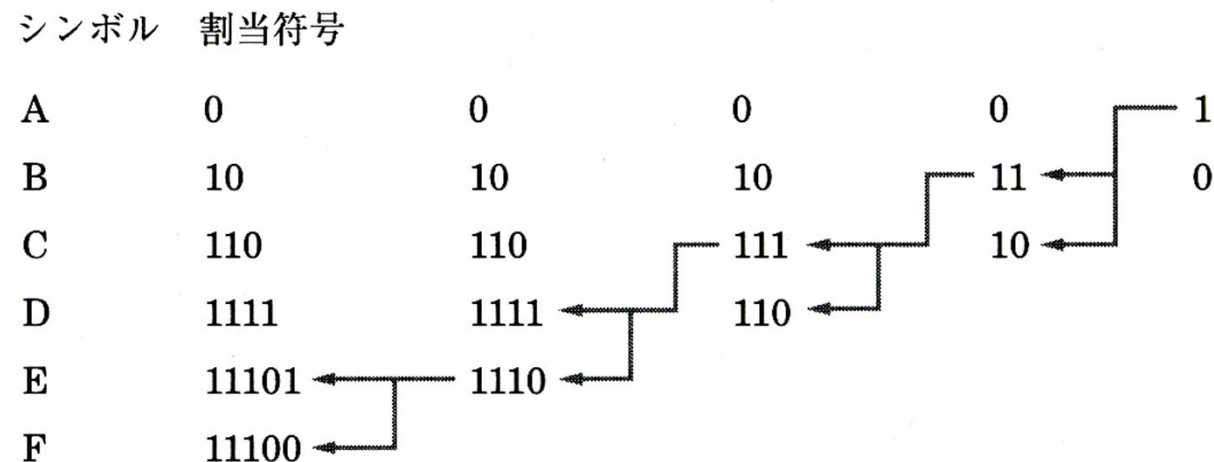
シンボル	発生確率							
A	0.4	0.4	0.4	0.4	0.4	0.6		
B	0.25	0.25	0.25	0.25	0.35	0.4		
C	0.15	0.15	0.2	0.2	0.25			
D	0.1	0.1	0.15	0.15				
E	0.06	0.1						
F	0.04							



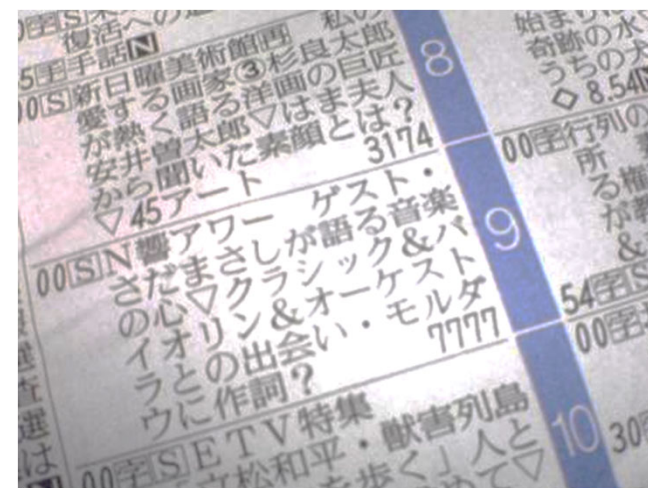
解答例

82

- ハフマンコードを用いなければ、3bit符号化が必要で、全部で 3×10 文字 = 30bit必要
- A(0), C(110), B(10), D(1111), A(0), E(11101), A(0), B(10), C(110), A(0) = 23bit
- よって、元の情報を77%に圧縮される。つまり圧縮率77%



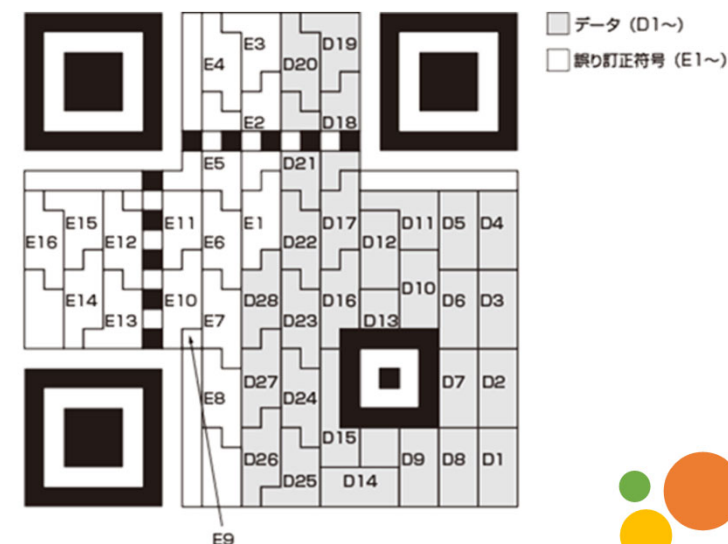
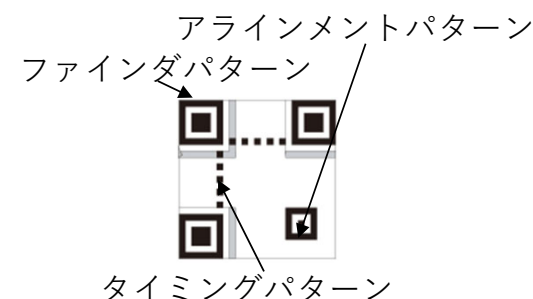
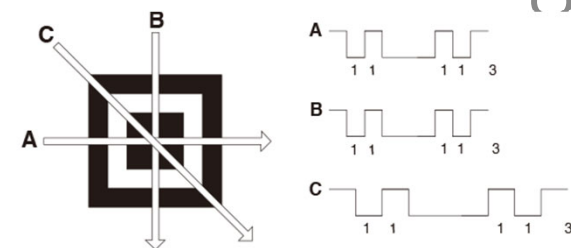
- 例えば、
 - 2009年6月1日NHK-Ech 21時0分から30分の予約は「1」
 - 2009年6月28日35ch23時26分から4分の予約は「902664」
 - 全ての日時、chがGコードで表現できるわけではない
- 予約録画に必要なデータ（日、時刻、録画時間、チャンネル）を分解・再構成し、最大8桁の数字列に圧縮
 - 放送日1～31日、開始時刻0時00分～23時59分、録画時間：5～480分、チャンネル：1～99、これを掛け合わせると2,099,196,000通り
 - 統計的分類に基づき開始時刻の自由度を分類
 - ナップザック詰め込み：
 - 開始時・分、録画時間分を15ビットに圧縮
 - サイファースブロック演算：
 - デコード過程で必要な情報が順次得られるようにする



QRコード(2次元バーコード)

84

- 21×21～73×73セルで構成
 - 最大数字1,167文字、英数字707文字、486バイトを表現
 - QRコードの3コーナーにあるファインダパターン
 - どの方向にスキャンしても白黒比が1:1:3:1:1となることを利用
 - 開発者はこの比率のパターンはどの言語でも出現が稀と主張している
 - アライメントパターンと呼ばれる小さなマーカを検出
 - タイミングパターンを利用してセルを検出
- エラー訂正（リードソロモン符号を利用）
 - 訂正レベルは4レベル
 - レベルLは約7%、レベルMは約15%、レベルQは約25%、レベルHは約30%が復元可能（汚れ対策）





演習問題（9）

85

- Processingを用いて次の簡易版1次元バーコードを読み取りなさい
 - バーコード画像の大きさは横幅が50から300で自由であり高さは問わない
 - 画像の回転は考えないが(考えてもよい)、横のサイズは変更可能とする
 - このバーコードが1つ、500x500の画像の中に埋め込まれている
 - その中に少なくとも0か1かが8桁以上含まれている
 - 0は短い黒と長い白、1は長い黒と短い白で構成される
 - 必ず最初は10の2桁から始まり、最後は1で終わるコードで、それ以外の部分を読み込むことができればよい
 - 次のサンプルコードを参考にしなさい
 - drawbcはバーコードを生成するコードである
 - 検出する部分は自由に設計してよい
 - 難しいと感じた学生向けに、ほとんどの部位を記述しているが、工夫のため変更してよい
 - `//コメント` のところに必要な処理(複数行であろう)を記入しなさい(基準点6点)
 - 単純にimageで貼り付けているが、回転、移動、拡大縮小なども含めて工夫した点があれば明記すること




```
void setup() {  
  size(500, 150);  
  background(255);  
  int ls = 8, ss = 4;  
  PImage bimg = drawbc("1010010101", ls, ss, 150);  
  image(bimg, 10, 0);  
  loadPixels();  
  int wb = 0, bw = 0;  
  for(int y = 1; y < 500; y++){  
    int pp = (int)red(pixels[y-1]), cp = (int)red(pixels[y]);  
    if((wb == 0) && (pp != 0) && (cp == 0)) wb = y;  
    if((wb > 0) && (pp == 0) && (cp != 0)) bw = y;  
    if((bw > 0) && (pp != 0) && (cp == 0)){  
      if((wb+y)/2 > bw) print("0");  
      if((wb+y)/2 < bw) print("1");  
      // このあたりを記述  
    }  
  }  
}
```

```
PImage drawbc(String mycode, int ls, int ss, int h) {  
  color blk = color(0,0,0);  
  color wht = color(255,255,255);  
  int[] nums = int(mycode.split(""));  
  int ll = ls+ss;  
  int[] cs = new int[ll*nums.length];  
  PImage bimg = createImage(nums.length*ll, h, RGB);  
  for (int i = 0; i < nums.length; i++)  
    for (int y = 0; y < h; y++)  
      for (int x = 0; x < ll; x++)  
        if (nums[i] == 0) {  
          if (x < ss) bimg.set(x+ll*i, y, blk);  
          else bimg.set(x+ll*i, y, wht);  
        } else {  
          if (x < ls) bimg.set(x+ll*i, y, blk);  
          else bimg.set(x+ll*i, y, wht);  
        }  
  return bimg;  
}
```

