画像処理

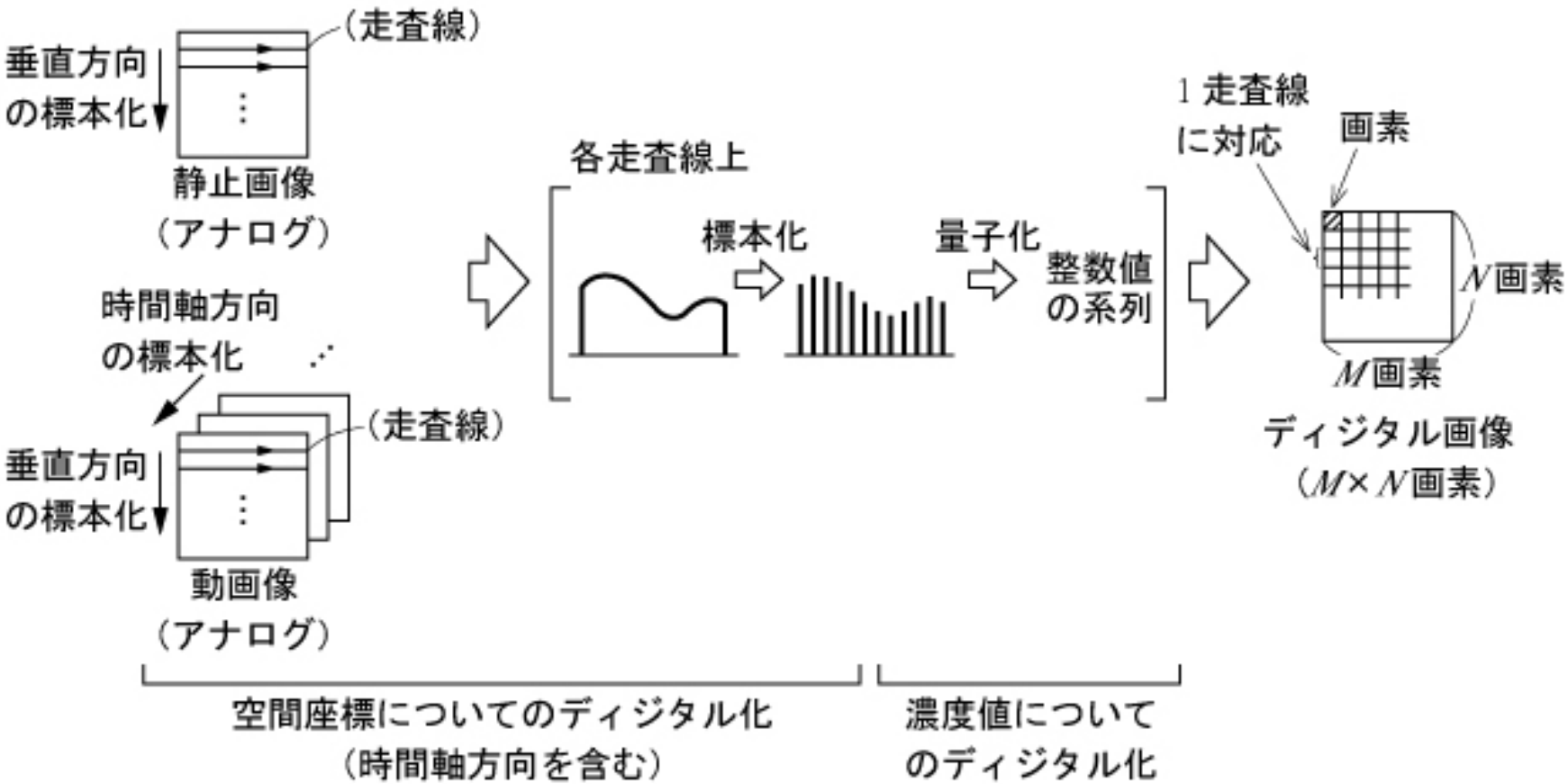
宮崎大学 工学部 情報システム工学科

３年後期
第 １ 回

本日の内容

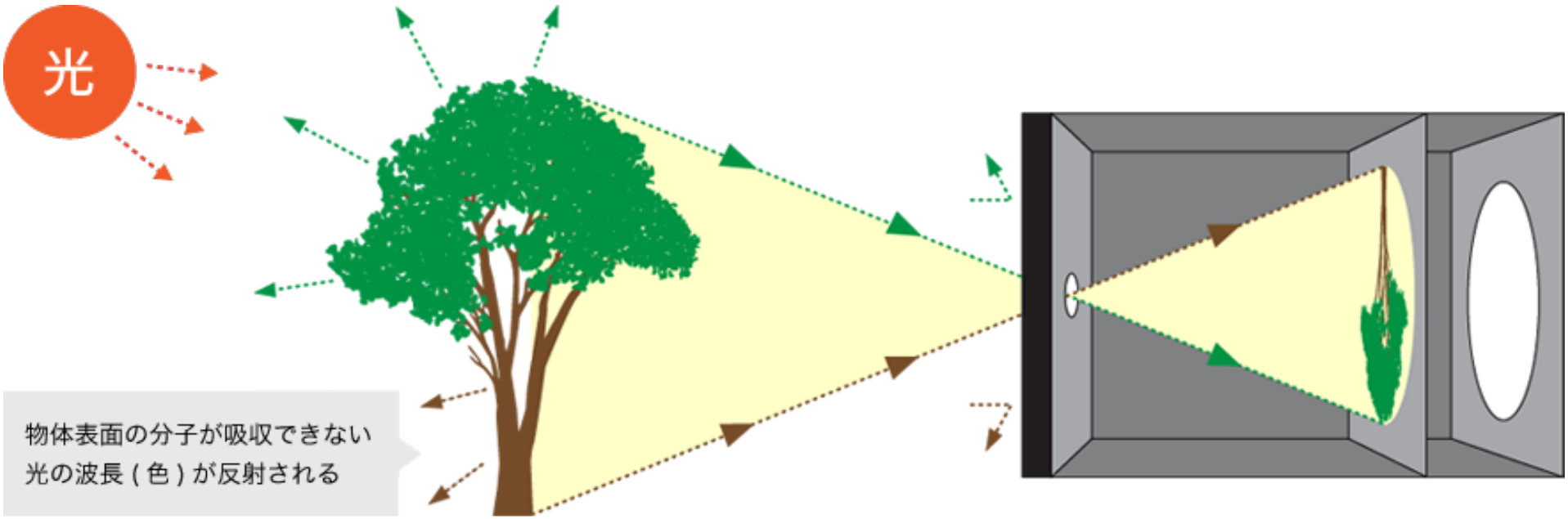
- 標本化，量子化
- 種々の画像
  - ２値，濃淡画像
  - カラー画像，表色系
  - 動画像，３次元画像
- 画像のデータ表現

デジタル画像

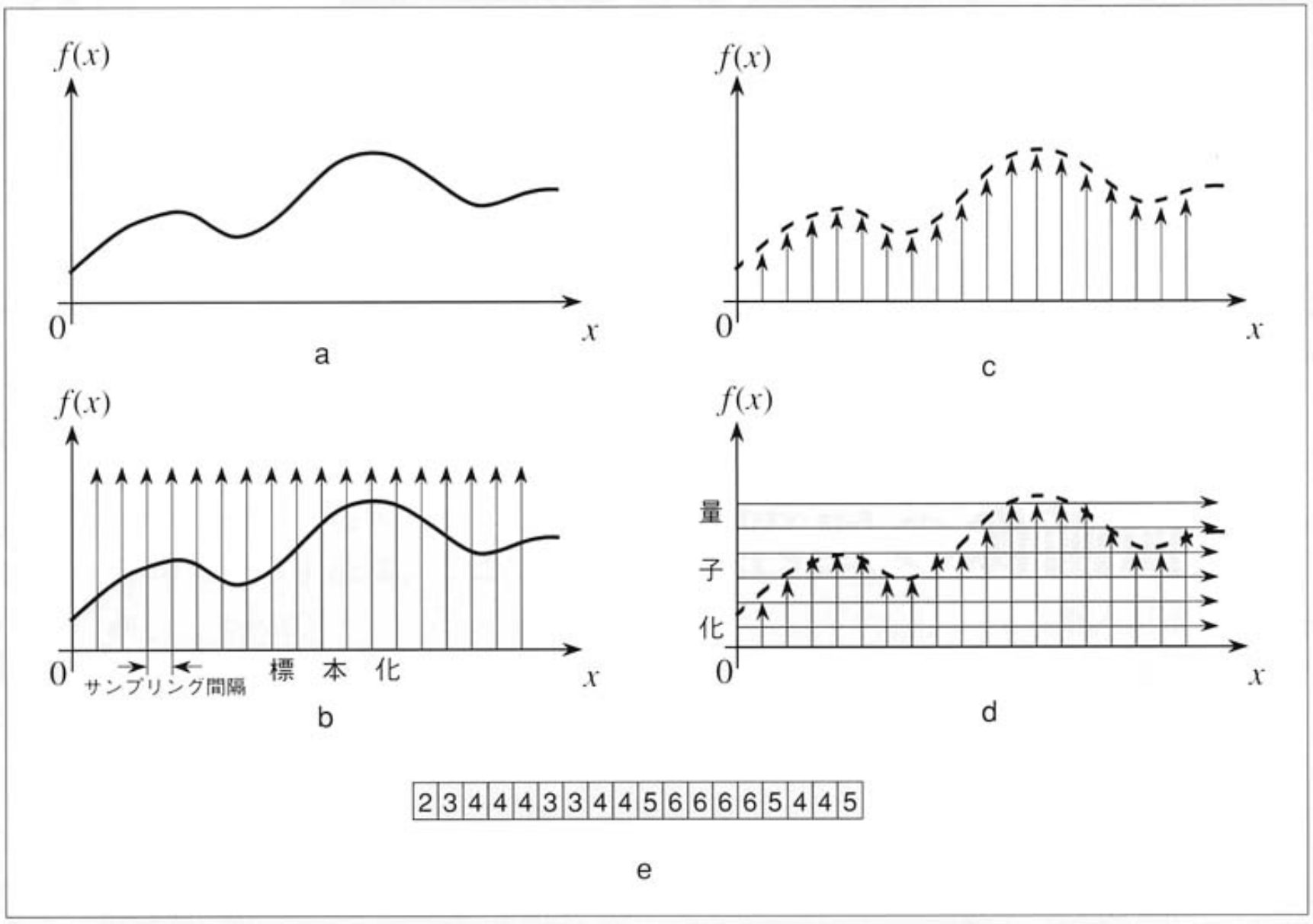


- ピンホールカメラ／レンズカメラにより撮影
- ラスタ走査により標本化
- 濃淡値の量子化によりデジタル化

ピンホールカメラモデル



標本化と量子化


f(x)
0
x
a
f(x)
0
x
c
f(x)
0
サンプリング間隔
標 本 化
x
b
f(x)
量
子
化
0
x
d
2 3 4 4 4 3 3 4 4 5 6 6 6 6 5 4 4 5
e

標本化

空間的時間的に連続した情報を離散的な点の集合に変換

点 ＝画素（ピクセル, *pixel, picture element*）

情報 ＝画素位置での明るさ（連続値）

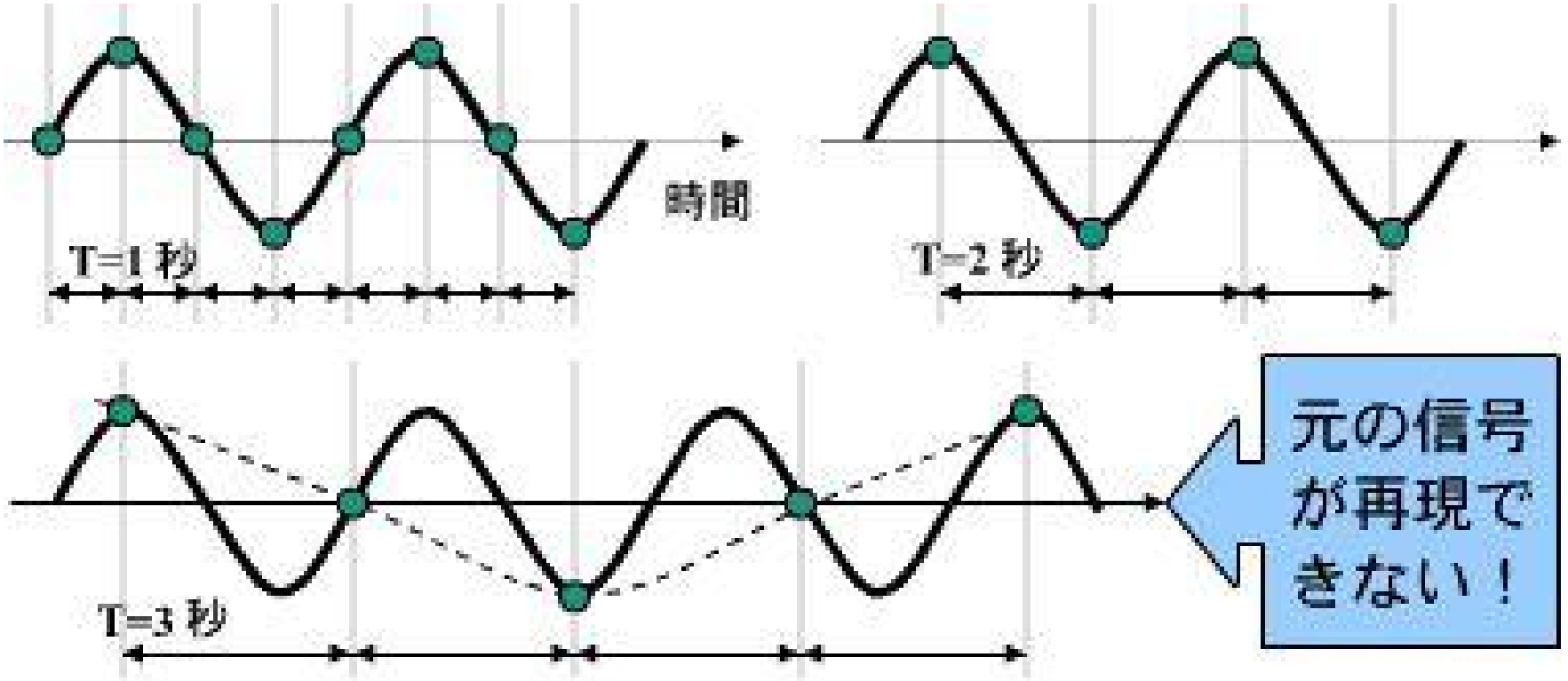
標本化定理

原信号の波長が λ 以上であるとき，
間隔 $\lambda/2$ 未満の離散個の信号の値から
もとの信号を完全に復元することができる．

⇒ 条件を満たす原信号については
標本化による誤差は生じない

参考：解像度，画像の大きさ

標本化定理



量子化

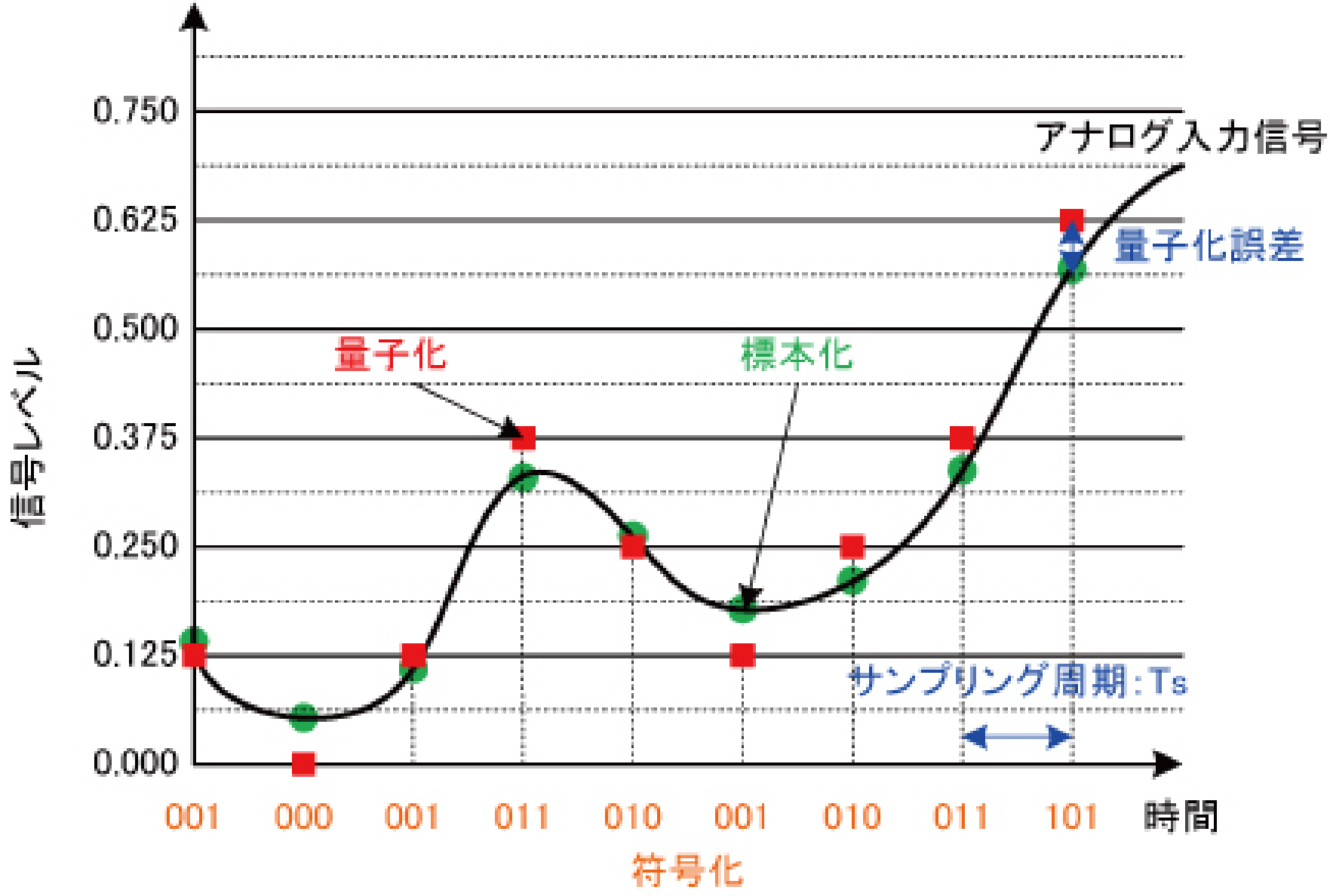
連続な値を離散的な値に変換

量子化誤差：量子化の際に生じる誤差

量子化レベル（階調）：離散値の段数

- 通常は 256 階調（8bit で表現）
- 高い精度の階調表現が必要な場合
  10~16bit で表現することもある

量子化と誤差


0.750
0.625
0.500
0.375
0.250
0.125
0.000
信号レベル
アナログ入力信号
量子化誤差
量子化
標本化
サンプリング周期：Ts
001
000
001
011
010
001
010
011
101
時間
符号化

解像度の違い

(a)　256×256 画素

(b)　128×128 画素

(c)　64×64 画素

(d)　32×32 画素

量子化レベルの違い

(a)　256(＝2^8) レベル

(b)　64(＝2^6) レベル

(c)　16(＝2^4) レベル

(d)　4(＝2^2) レベル

(e)　2(＝2^1) レベル

各種の静止画

２値画像

- 量子化レベルが２（0＝白と１＝黒の２値）
- 文字や図形の画像，マスク画像などで使用

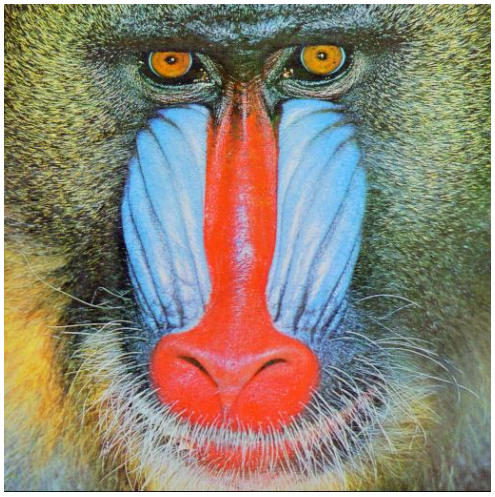
濃淡画像

- 量子化レベルが多段階（256~65536 階調）
- 一般情景，航空写真，Ｘ線写真などで使用

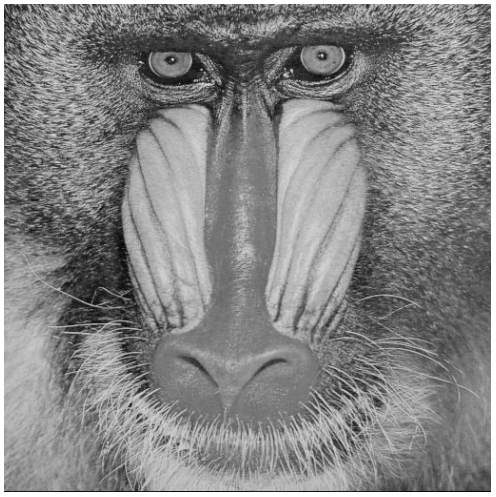
カラー画像

- 量子化レベルが多段階かつベクトル形式
- 色彩を含んだ一般情景を表現するために使用

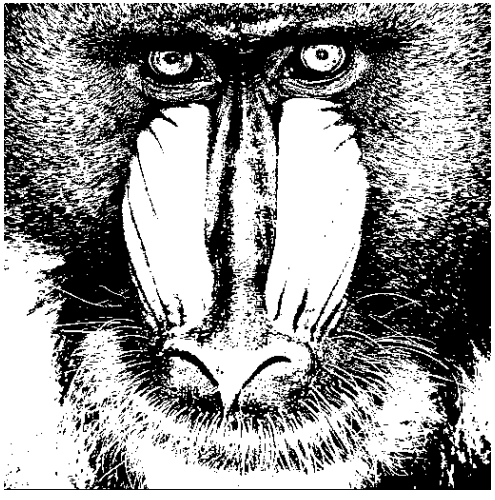
各種の静止画の例



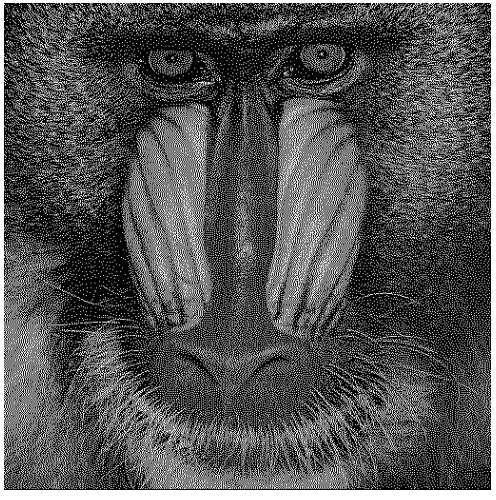
カラー



濃淡



２値（閾値）



２値（ディザ）

２値（マスク）

表色系

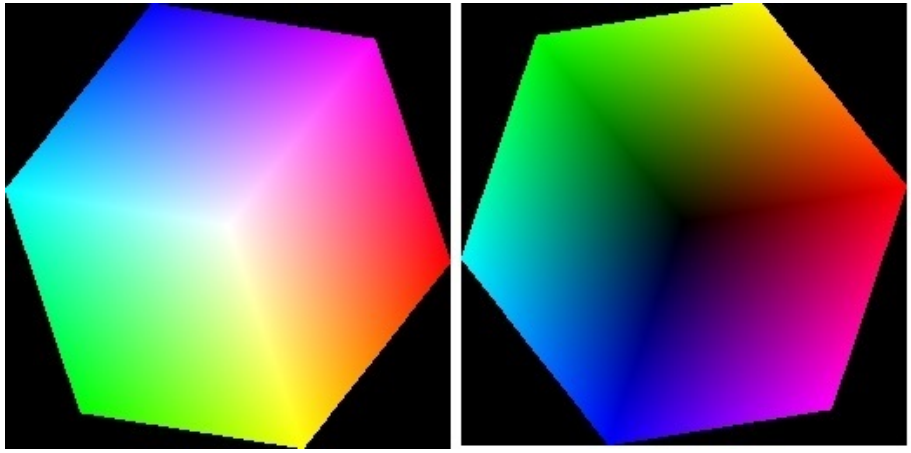
色を表す体系

光：各種の波長をもった電磁波

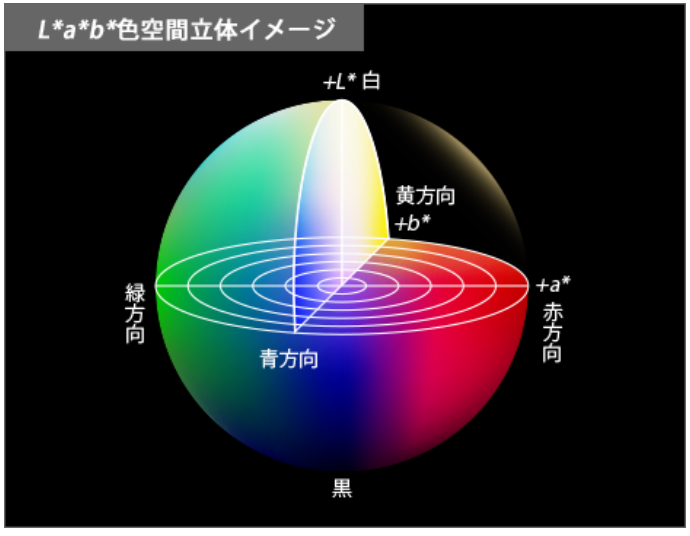
色：光のもつ波長の違いが観測されたもの

1. ３刺激値による表現
   独立な３つ色の線形和で表現 (RGB ／ XYZ)
2. 色差による表現
   人間の感じる色の違いを定量化
   (CIE LAB ／ CIE LUV/YIQ)
3. 色相，彩度，明度による表現
   人間の感じる色を反映 (HSV/HSI/HSL)

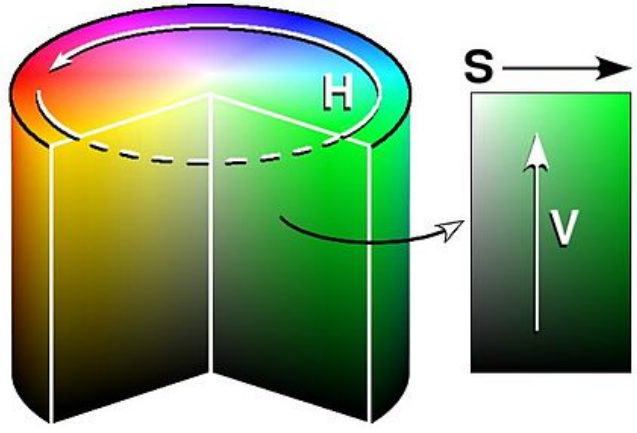
表色系の例



RGB



LAB



HSV

その他の画像

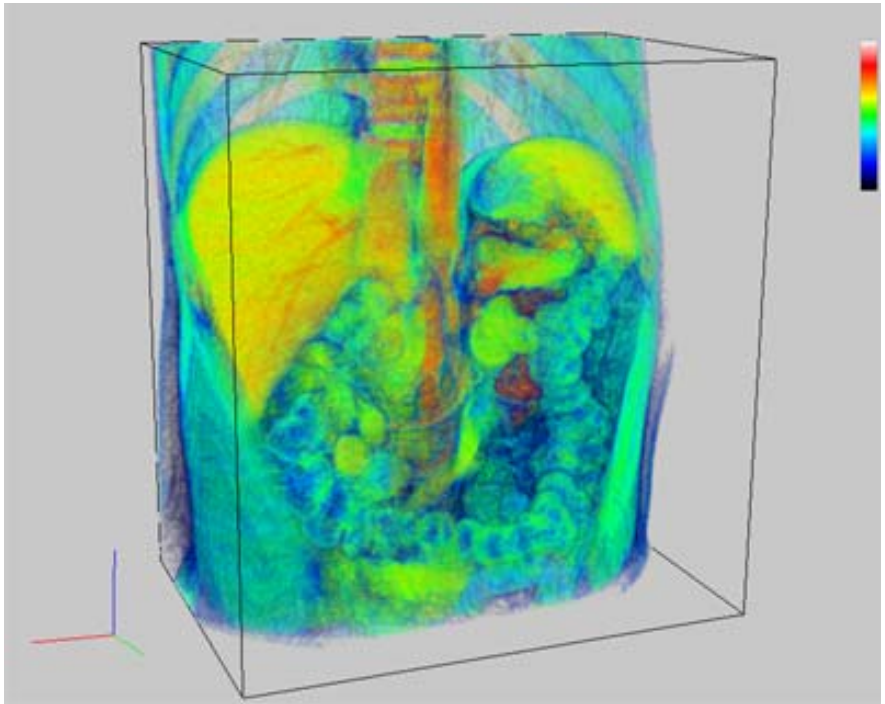
動画像

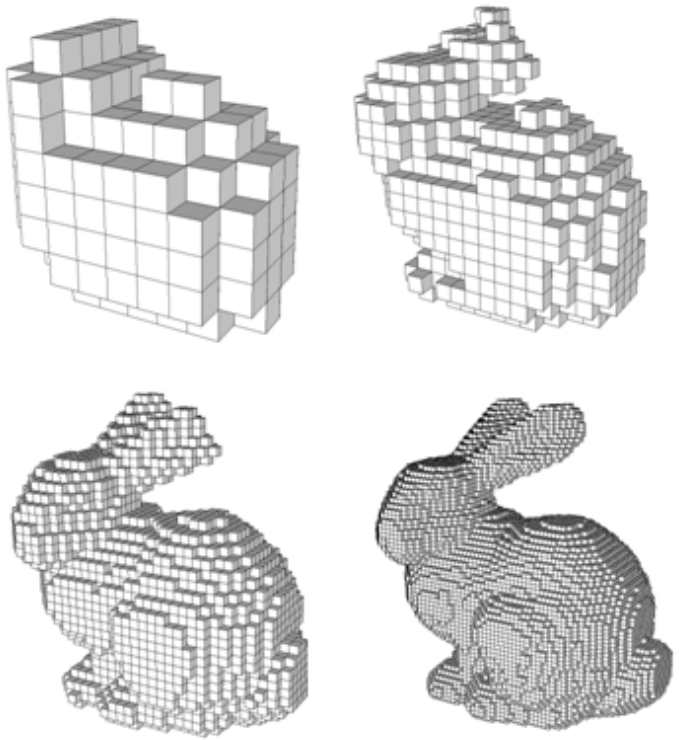
複数の静止画を時間順に並べたもの

３次元画像（ボリュームデータ）

３次元空間での位置毎の観測量を表すために，３次元状にデータを並べたもの

画素⇒ ボクセル（*voxel*）





ファイルフォーマット

静止画

- 非圧縮／可逆圧縮／不可逆圧縮がある
- 走査順に画素値を並べるものと，
  カラーパレットを使うものがある

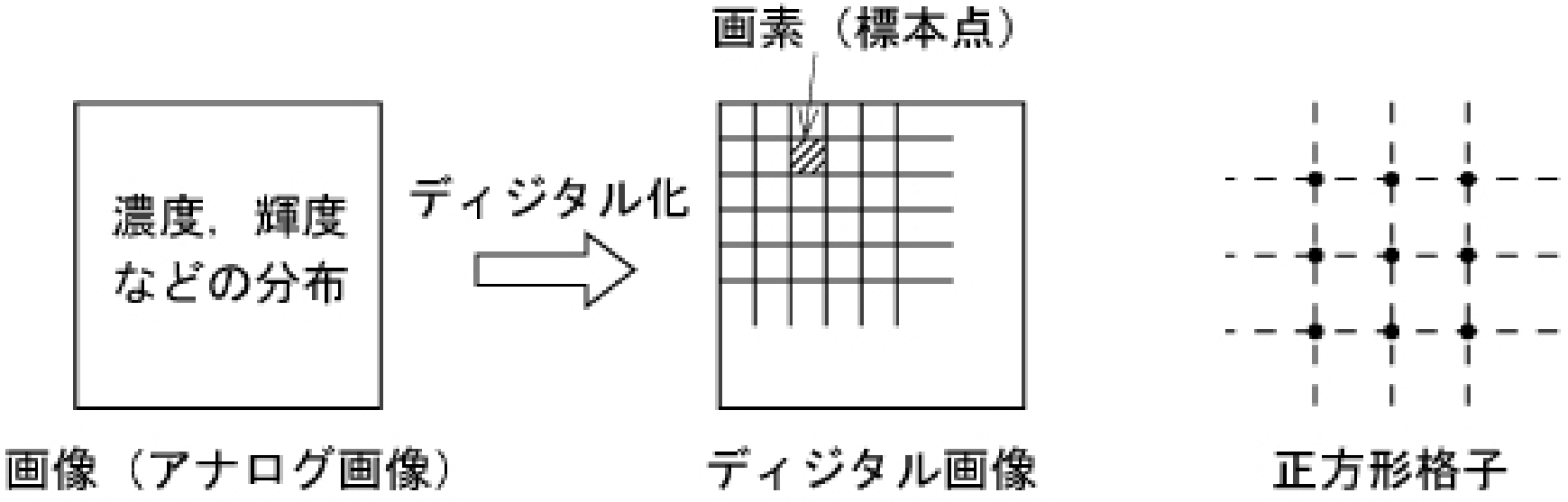
例：BMP, PBMPLUS, GIF, PNG, JPEG

動画

- データ量が大きくなるので，ほとんどが不可逆圧縮
- フレーム間の類似性を使って圧縮率を高めている

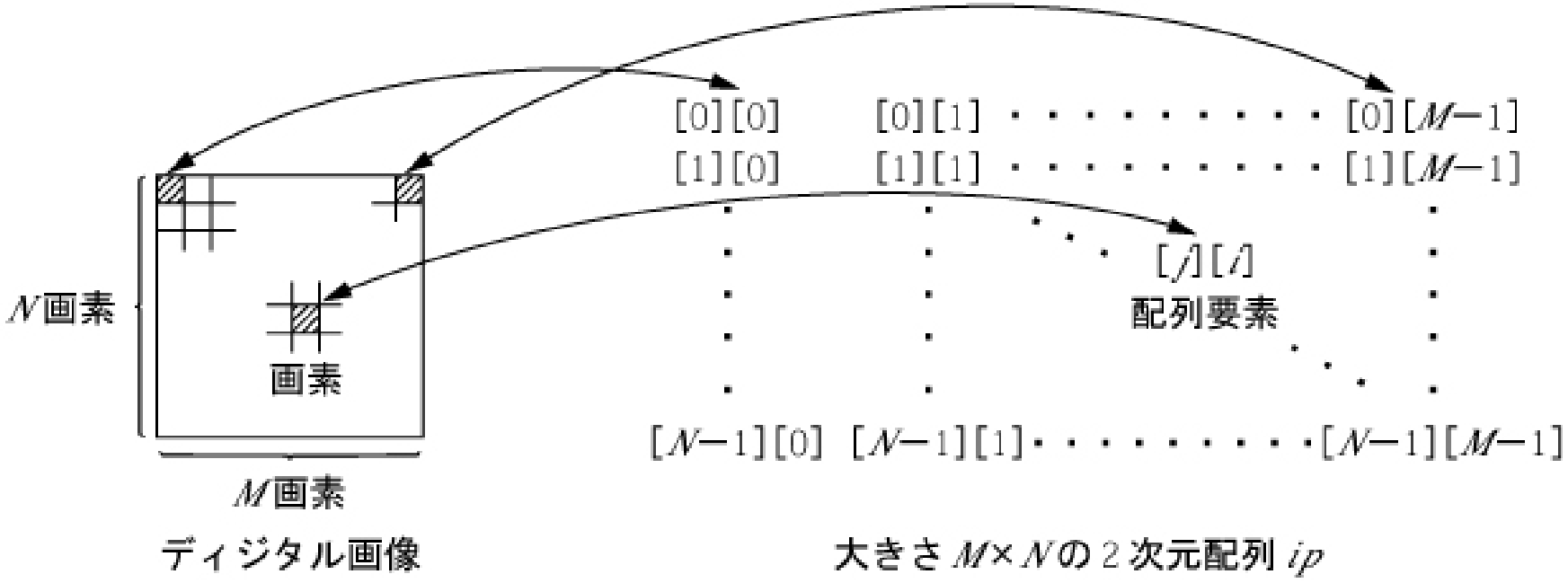
例：MotionJPEG, MPEG1, MPEG2, MPEG4, WMV, AVI, MOV

２次元配列による画像の表現



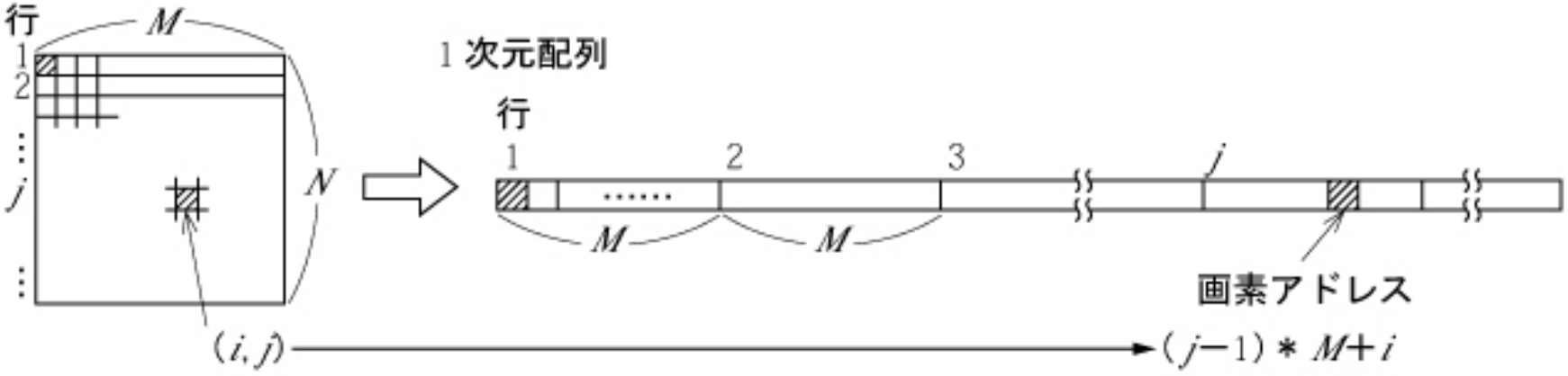
標本化，量子化によりデジタル表現された画像は，
２次元状に分布したデータとみなせる．
⇒ 計算機内部では，２次元配列として扱える．

２次元配列による画像のデータ表現



- 左上が原点．右向きに x 軸．下向きに y 軸．
- 座標は 0 から始まる．画素数 M の時，最終は M-1
- プログラム上の表記では，[y][x] の順番

1 次元配列による画像のデータ表現



```c
/*************************************************************
*
*   Calculate Average Pixel Value for Gray Image
*
*   filename    :       average_gray.c
*   writer      :       M.Mukunoki
*   date        :       2016/08/25
*
*   compile     :
*    gcc -I. average_gray.c kumi3.c -o average_gray
*
*************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <kumi3.h>

/**** Defines ****/

/**** Macros ****/

/**** Typedefs ****/
typedef unsigned char uchar;

/**** External Variables ****/

/**** Prototype Declaration ****/
int main(int argc, char *argv[]);
int usage(char *command);
```

```c
int average_gray(K_IMAGE *img, float *ave);

/**** Routines ****/
int main(int argc, char *argv[])
{
    K_IMAGE     *inp_img;
    char        *inp_fname      = K_STDFILE;

    int         c;
    int         errflg = 0;
    float       ave;

    extern int  optind;
    extern char *optarg;

    while((c = getopt(argc, argv, "i:h")) != EOF)
    {
        switch(c)
        {
        case 'i':
            inp_fname = optarg;
            break;
        case 'h':
            errflg++;
            break;
        default:
            errflg++;
            break;
        }
```

```
    if (errflg)
    {
        usage(argv[0]);
        exit(1);
    }
}/* end of while */
if (optind < argc)
{
    inp_fname = argv[optind++];
}
if (optind != argc)
{
    usage(argv[0]);
    exit(1);
}

if ((inp_img = k_open(inp_fname)) == NULL)
{
    return(-1);
}

if (k_pixeltype(inp_img) != K_UCHAR)
{
    fprintf(stderr, "Pixeltype of input image must be K_UCHAR!!\n");
    exit(1);
}

average_gray(inp_img, &ave);
printf("%f\n", ave);
```

```
    k_close(inp_img);

    exit(0);
}

int usage(char  *command)
{
    fprintf(stderr, "Calculate Average Pixel Value for Gray Image.\n");
    fprintf(stderr, "Usage: %s [-h] ", command);
    fprintf(stderr, "[[-i] input_file\n");
    fprintf(stderr, "\t -h : help\n");
    fprintf(stderr, "\t -i : input_file (you can omit '-i')\n");

    return(0);
}

int average_gray(K_IMAGE *img, float *ave)
{
    int         sum = 0;
    uchar       **iptr = (uchar **)(k_data(img)[0]);

    for(int y = 0; y < k_ysize(img); y++)
    {
        for(int x = 0; x < k_xsize(img); x++)
        {
            sum += iptr[y][x];
        }
    }
```

```
    *ave = (float)sum/(k_xsize(img) * k_ysize(img));

    return 0;
}
```

画像入出力ライブラリ KUMI3

- C 言語で画像の入出力，画素値操作を行うライブラリ
- 独自の kumi 形式画像の他に， ppm, pgm, pbm, jpg, png 形式が扱える
- 1 画素を記憶する形式 (pixeltype) として以下が使える

  `K_UCHAR, K_USHORT, K_INT, K_FLOAT, K_DOUBLE, K_BIT`
- マルチスペクトル画像が使える (多重度 multi)

画像入出力ライブラリ KUMI3

画像を記憶する構造体 K_IMAGE

```
typedef struct
{
  K_HEAD *header;    // header information
  int    pixelbyte; // number of bytes for a pixel
  void   ***data;    // pointer to the buffer for images
} K_IMAGE;
```

画像入出力ライブラリ KUMI3

画像の情報を記憶する構造体 K_HEAD

```
typedef struct
{
  int stype;   // store type
  int xsize;   // horizontal size of the image
  int ysize;   // vertical size of the image
  int pixeltype; // pixel type
  int ebit;    // effective bit
  int imgtype; // image type (dummy)
  int multi;   // multiply of frames
  int order;   // number of frames
} K_HEAD;
```

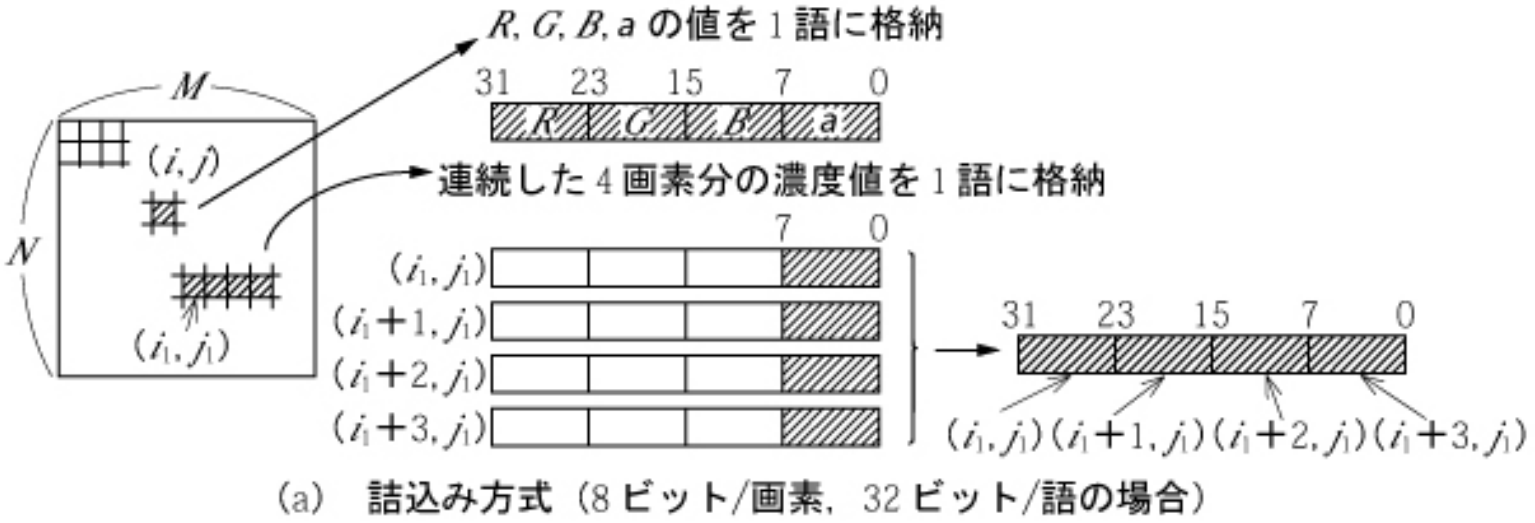
画像入出力ライブラリ KUMI3

主な関数・マクロ

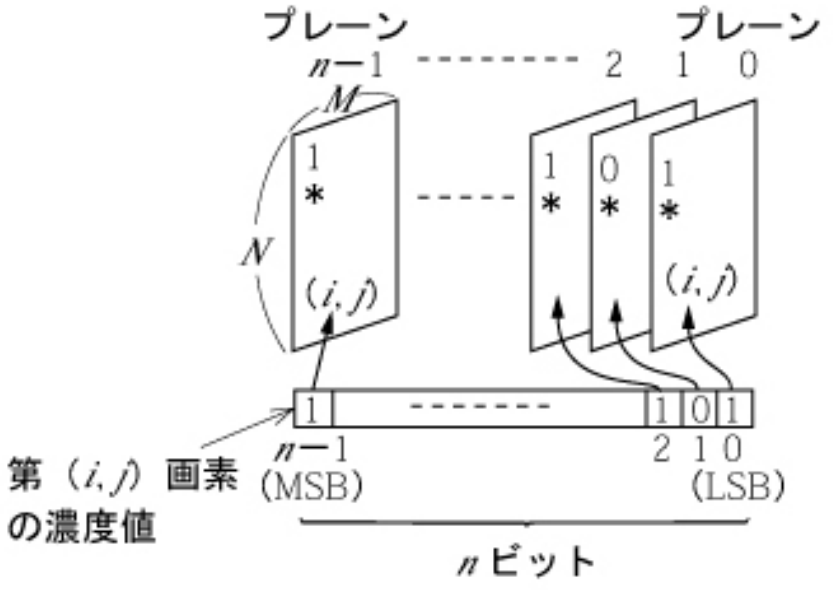
```
K_IMAGE *k_open(char *fname)          // 画像の読み込み
K_IMAGE *k_create(K_HEAD *header)    // 新たな画像の作成
int k_write(K_IMAGE *img, char *fname); // 画像の書き込み
int k_close(K_IMAGE *img);           // 画像の破棄

k_xsize(image)      // 画像の横の画素数
k_ysize(image)      // 画像の縦の画素数
k_pixeltype(image) // 1 画素を表すためのデータ形式
k_multi(image)      // 画像の多重度
k_header(image)     // 画像情報 (K_HEAD) へのポインタ
k_data(image)       // 画像領域 (void ***) へのポインタ
```

カラー画像のデータ表現



(a) 詰込み方式（8 ビット/画素，32 ビット/語の場合）



(b) ビット・プレーン方式

```c
/*************************************************************
*
*   Calculate Average Pixel Value for RGB Image
*
*   filename    :       average_rgb.c
*   writer      :       M.Mukunoki
*   date        :       2016/08/25
*
*   compile     :
*    gcc -O3 -funroll-loops average_rgb.c kumi3.c -o average_rgb
*
*************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <kumi3.h>

/**** Defines ****/

/**** Macros ****/

/**** Typedefs ****/
typedef unsigned char uchar;

/**** External Variables ****/

/**** Prototype Declaration ****/
int main(int argc, char *argv[]);
int usage(char *command);
```

```
int average_rgb(K_IMAGE *img, float *ave);

/**** Routines ****/
int main(int argc, char *argv[])
{
    K_IMAGE     *inp_img;
    char        *inp_fname      = K_STDFILE;

    int         c;
    int         errflg = 0;

    extern int  optind;
    extern char *optarg;

    while((c = getopt(argc, argv, "i:h")) != EOF)
    {
        switch(c)
        {
        case 'i':
            inp_fname = optarg;
            break;
        case 'h':
            errflg++;
            break;
        default:
            errflg++;
            break;
        }
        if (errflg)
```

```
    {
        usage(argv[0]);
        exit(1);
    }
}/* end of while */
if (optind < argc)
{
    inp_fname = argv[optind++];
}
if (optind != argc)
{
    usage(argv[0]);
    exit(1);
}

if ((inp_img = k_open(inp_fname)) == NULL)
{
    return(-1);
}

if (k_pixeltype(inp_img) != K_UCHAR)
{
    fprintf(stderr, "Pixeltype of input image must be K_UCHAR!!\n");
    exit(1);
}

float       ave[k_multi(inp_img)];

average_rgb(inp_img, ave);
```

```
    for(int m = 0; m < k_multi(inp_img); m++)
    {
        printf("%d %f\n", m, ave[m]);
    }

    k_close(inp_img);

    exit(0);
}

int usage(char  *command)
{
    fprintf(stderr, "Calculate Average Pixel Value for RGB Image.\n");
    fprintf(stderr, "Usage: %s [-h] ", command);
    fprintf(stderr, "[[-i] input_file\n");
    fprintf(stderr, "\t -h : help\n");
    fprintf(stderr, "\t -i : input_file (you can omit '-i')\n");

    return(0);
}

int average_rgb(K_IMAGE *img, float *ave)
{
    uchar       ***iptr = (uchar ***)k_data(img);

    for(int m = 0; m < k_multi(img); m++)
    {
        int             sum = 0;
        for(int y = 0; y < k_ysize(img); y++)
```

```
        {
            for(int x = 0; x < k_xsize(img); x++)
            {
                sum += iptr[m][y][x];
            }
        }
        ave[m] = (float)sum/(k_xsize(img) * k_ysize(img));
    }

    return 0;
}
```

```c
/*****************************************************************
*
*   Copy Image
*
*   writer  :   M.Mukunoki
*   date    :   2016/12/08
*   compile :
*    gcc -I. copy_img.c kumi3.c -o copy_img
*   execute :
*    ./copy_img sample01.pgm out01.pgm
*
*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <kumi3.h>

/**** Defines ****/
#define DEF_THRES       128

/**** Macros ****/

/**** Typedefs ****/
typedef unsigned char uchar;

/**** External Variables ****/

/**** Prototype Declaration ****/
int main(int argc, char *argv[]);
```

```
int usage(char *command);
int copy_img(K_IMAGE *inp_img, K_IMAGE *out_img);

/**** Routines ****/
int main(int argc, char *argv[])
{
    K_IMAGE     *inp_img, *out_img;
    char        *inp_fname      = K_STDFILE;
    char        *out_fname      = K_STDFILE;

    int         c;
    int         errflg = 0;
    int         thres = DEF_THRES;

    extern int  optind;
    extern char *optarg;

    while((c = getopt(argc, argv, "i:o:Pt:h")) != EOF)
    {
        switch(c)
        {
        case 'i':
            inp_fname = optarg;
            break;
        case 'o':
            out_fname = optarg;
            break;
        case 't':
            thres = atoi(optarg);
```

```
            break;
        case 'P':
            out_fname = K_STDPNM;
            break;
        case 'h':
            errflg++;
            break;
        default:
            errflg++;
            break;
        }
        if (errflg)
        {
            usage(argv[0]);
            exit(1);
        }
    }/* end of while */
    if (optind < argc)
    {
        inp_fname = argv[optind++];
    }
    if (optind < argc)
    {
        out_fname = argv[optind++];
    }
    if (optind != argc)
    {
        usage(argv[0]);
        exit(1);
```

```
    }

    if ((inp_img = k_open(inp_fname)) == NULL)
    {
        return(-1);
    }
    if (k_pixeltype(inp_img) != K_UCHAR)
    {
        fprintf(stderr, "Pixeltype of input image must be K_UCHAR!!\n");
        exit(1);
    }

    if ((out_img = k_create(k_header(inp_img))) == NULL)
    {
        return(-1);
    }

    copy_img(inp_img, out_img);

    k_write(out_img, out_fname);

    k_close(inp_img);
    k_close(out_img);

    return 0;
}

int usage(char  *command)
{
```

```
    fprintf(stderr, "Copy Image.\n");
    fprintf(stderr, "Usage: %s [-hP] [-t thres] ", command);
    fprintf(stderr, "[[-i] input_file [[-o] output_file]]\n");
    fprintf(stderr, "\t -h : help\n");
    fprintf(stderr, "\t -t : threshold value\n");
    fprintf(stderr, "\t -P : output with NETPBM format\n");
    fprintf(stderr, "\t -i : input_file (you can omit '-i')\n");
    fprintf(stderr, "\t -o : output_file (you can omit '-o')\n");

    return(0);
}

int copy_img(K_IMAGE *inp_img, K_IMAGE *out_img)
{
    uchar       ***iptr = (uchar ***)k_data(inp_img);
    uchar       ***optr = (uchar ***)k_data(out_img);

    for(int m = 0; m < k_multi(inp_img); m++)
    {
        for(int y = 0; y < k_ysize(inp_img); y++)
        {
            for(int x = 0; x < k_xsize(inp_img); x++)
            {
                optr[m][y][x] = iptr[m][y][x];
            }
        }
    }

    return 0;
```

```
}
```