JPEG

BY CHIA-CHUN, LIN

DIAGRAM

RGB To YCbCr

Blocking 8 x 8

DCT

Quantization

Quantization table

AC

DC

DPCM

Y、Cb、Cr

huffman

Huffman Tree table

zigzag

RLE

Y、Cb、Cr

RGB TO YCBCR

$$Y=0.299*R + 0.587*G + 0.114*B$$

$$Cb=0.564*(B-Y)$$

$$Cr=0.713*(R-Y)$$

YCBCR TO RGB

$$R=Y + 1.402*Cr;$$

$$G=Y - 0.344*Cb - 0.714*Cr$$

$$B=Y + 1.722*Cb$$

DCT

```
209  YCbCr** DCT(YCbCr **input, int Nh, int Nw, double **cosi){
210      YCbCr **output = YCbCr_2D(Nh, Nw);
211      int m,n,u,v,x,y;
212      double *z,c,temp;
213      z=calloc(3, sizeof(double));
214      for(m=0;m<Nh;m+=8){
215          for(n=0;n<Nw;n+=8){
216              for(u=0;u<8;u++){
217                  for(v=0;v<8;v++){
218                      c=1.0;
219                      z[0]=0.0;
220                      z[1]=0.0;
221                      z[2]=0.0;
222                      for(x=0;x<8;x++){
223                          for(y=0;y<8;y++){
224                              temp = cosi[x][u] * cosi[y][v];
225                              z[0]+=(input[m+x][n+y].Y-128) * temp;
226                              z[1]+=input[m+x][n+y].Cb * temp;
227                              z[2]+=input[m+x][n+y].Cr * temp;
228                          }
229                      }
230                      if(u==0) c/=sqrt(2);
231                      if(v==0) c/=sqrt(2);
232                      output[m+u][n+v].Y = c * z[0] / 4;
233                      output[m+u][n+v].Cb = c * z[1] / 4;
234                      output[m+u][n+v].Cr = c * z[2] / 4;
235                  }
236              }
237          }
238      }
239      free(z);
240      return output;
241  }
```

QUATIZATION

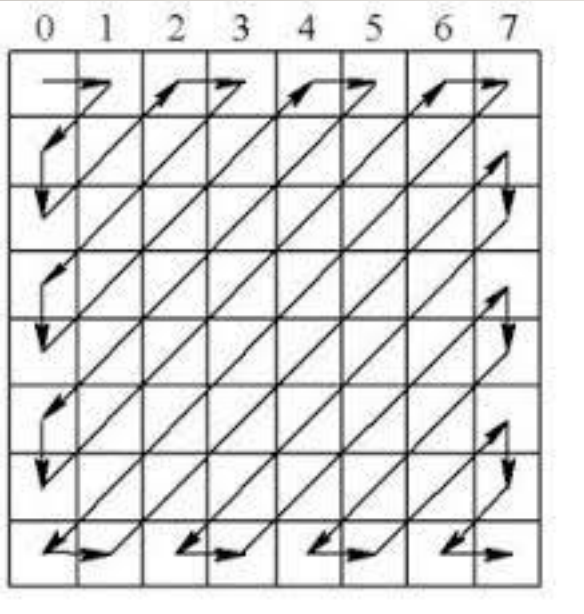
```
7   // quantization table (luminance)
8   const int Q_lumi[8][8]={
9       16,  11,  10,  16,  24,  40,  51,  61,
10      12,  12,  14,  19,  26,  58,  60,  55,
11      14,  13,  16,  24,  40,  57,  69,  56,
12      14,  17,  22,  29,  51,  87,  80,  62,
13      18,  22,  37,  56,  68, 109, 103,  77,
14      24,  35,  55,  64,  81, 104, 113,  92,
15      49,  64,  78,  87, 103, 121, 120, 101,
16      72,  92,  95,  98, 112, 100, 103,  99,
17  };
```

```
// quantization table (chorminance)
const int Q_chorm[8][8]={
    17,  18,  24,  47,  99,  99,  99,  99,
    18,  21,  26,  66,  99,  99,  99,  99,
    24,  26,  56,  99,  99,  99,  99,  99,
    47,  66,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
};
```

DPCM

```
347  void DPCM(YCbCr **input, int Nh, int Nw, int *DC_Y, int *DC_Cb, int *DC_Cr){
348      int i,j,index=0;
349      // do in every row(DC)
350      for(i=0;i<Nh;i+=8){
351          for(j=Nw-8;j>0;j-=8){
352              input[i][j].Y -= input[i][j-8].Y;
353              input[i][j].Cb -= input[i][j-8].Cb;
354              input[i][j].Cr -= input[i][j-8].Cr;
355          }
356      }
357      // do in first column(DC)
358      for(i=Nh-8;i>0;i-=8){
359          input[i][0].Y -= input[i-8][0].Y;
360          input[i][0].Cb -= input[i-8][0].Cb;
361          input[i][0].Cr -= input[i-8][0].Cr;
362      }
363      // get DC
364      for(i=0;i<Nh/8;i++){
365          for(j=0;j<Nw/8;j++){
366              DC_Y[index] = input[i*8][j*8].Y;
367              DC_Cb[index] = input[i*8][j*8].Cb;
368              DC_Cr[index] = input[i*8][j*8].Cr;
369              index++;
370          }
371      }
372  }
```

ZIGZAG




```
const int row[64]={
    0,0,1,2,1,0,0,1,
    2,3,4,3,2,1,0,0,
    1,2,3,4,5,6,5,4,
    3,2,1,0,0,1,2,3,
    4,5,6,7,7,6,5,4,
    3,2,1,2,3,4,5,6,
    7,7,6,5,4,3,4,5,
    6,7,7,6,5,6,7,7};
```

```
const int col[64]={
    0,1,0,0,1,2,3,2,
    1,0,0,1,2,3,4,5,
    4,3,2,1,0,0,1,2,
    3,4,5,6,7,7,6,5,
    4,3,2,1,0,1,2,3,
    4,5,6,7,7,6,5,4,
    3,2,3,4,5,6,7,7,
    6,5,4,5,6,7,6,7};
```

```
374  void zigzag(YCbCr **input, int Nh, int Nw, int *AC_Y, int *AC_Cb, int *AC_Cr){
375      int m,n,i,index=0;
376      for(m=0;m<Nh;m+=8){
377          for(n=0;n<Nw;n+=8){
378              for(i=1;i<64;i++){
379                  AC_Y[index]=(int)input[m+row[i]][n+col[i]].Y;
380                  AC_Cb[index]=(int)input[m+row[i]][n+col[i]].Cb;
381                  AC_Cr[index]=(int)input[m+row[i]][n+col[i]].Cr;
382                  index++;
383              }
384          }
385      }
386  }
```

RLE

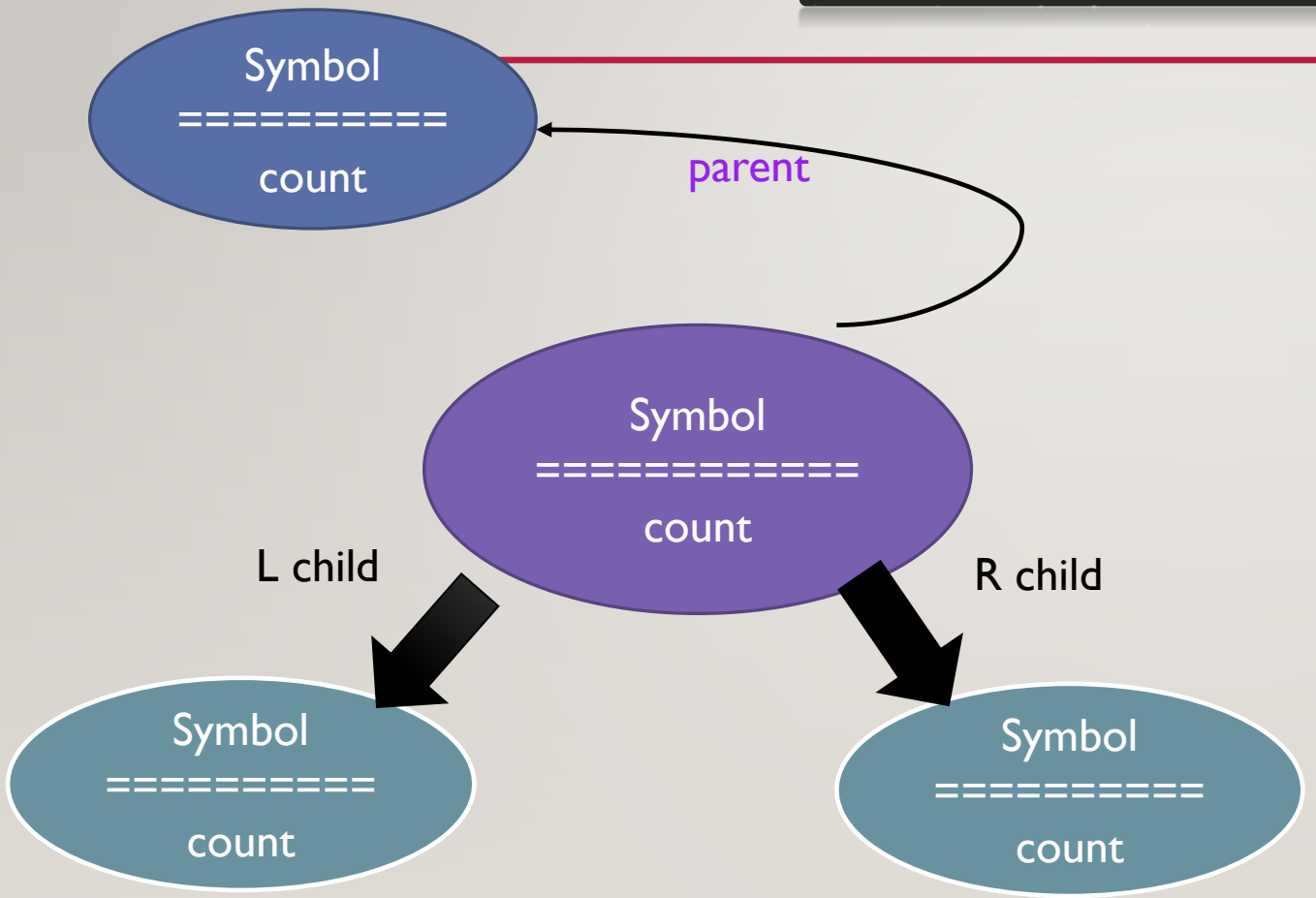
```
428  int* RLE(int *input, int size, int *rle_size){
429      int i=0,count=0,j=0;
430      int *temp, m;
431      temp = malloc(2*size*sizeof(int));
432      for(m=0;m<size;m+=63){
433          for(i=0;i<63;i++){
434              if(input[m+i]==0){
435                  count++;
436              }
437              else{
438                  temp[j]=count;
439                  temp[j+1]=input[m+i];
440                  count=0;
441                  j+=2;
442              }
443          }
444          temp[j]=0;
445          temp[j+1]=0;
446          count=0;
447          j+=2;
448      }
449      *rle_size = j;
450      int *output;
451      output=calloc(j,sizeof(int));
452      for(i=0;i<j;i++){
453          output[i]=temp[i];
454      }
455      free(temp);
456      return output;
457  }
```

GET__INFO FUNC.

```
481  int getinfo(int *input, int rle_size, int *output){
482      int i,j,legal,amount=0;
483      for(i=0;i<rle_size;i++){
484          legal=0;
485          for(j=0;j<2*amount;j+=2){
486              if(input[i]==output[j]){
487                  output[j+1]++;
488                  legal=1;
489                  break;
490              }
491          }
492          if(legal==0){
493              output[2*amount]=input[i];
494              output[2*amount+1]=1;
495              amount++;
496          }
497      }
498      return amount;
499  }
```

HUFFMAN CODE

```
void Huffman(int *rle_data, int rle_size, char *filename){
```




```
86  typedef struct treenode{
87      int symbol;
88      int count;
89      int parent;
90      int lchild,rchild;
91  } node;
92
93  typedef struct _codebook{
94      int bit[200];
95      int symbol;
96      int num;
97  } codebook;
```

```
559     int min1,min2,p1,p2;
560     for(i=0;i<amount-1;i++){
561         min1=min2=10000000;
562         for(j=0;j<amount+i;j++){
563             if(node[j].count<min1 && node[j].parent==-1){
564                 min2=min1;
565                 p2=p1;
566                 min1=node[j].count;
567                 p1=j;
568             }
569             else if(node[j].count<min2 && node[j].parent==-1){
570                 min2=node[j].count;
571                 p2=j;
572             }
573         }
574         node[p1].parent=amount+i;
575         node[p2].parent=amount+i;
576         node[amount+i].parent=-1;
577         node[amount+i].count=min1+min2;
578         node[amount+i].lchild=p1;
579         node[amount+i].rchild=p2;
580     }
581     // output table sheet
582     codebook *book=malloc(amount*sizeof(book));
583
584     for(i=0;i<amount;i++){
585         book[i].symbol = node[i].symbol;
586     }
587
```

```
HuffmanCode(node, book, amount, filename);
```

Initialize!!!

```
    // init
    for(i=0;i<2*amount-1;i++){
        node[i].symbol=0;
        node[i].count=0;
        node[i].lchild=-1;
        node[i].rchild=-1;
        node[i].parent=-1;
    }
```

Put data in

```
for(i=0;i<amount;i++){
    node[i].symbol=temp[2*i];
    node[i].count=temp[2*i+1];
}
```

HUFFMAN TREE

```
501  void HuffmanCode(node *node, codebook *book, int amount, char *filename){
502      FILE *fp;
503      fp=fopen(filename,"w+");
504
505      int temp[amount];
506      int index,start;
507      int i,j,c,p;
508
509      for(i=0;i<amount;i++){
510          start=amount-1;
511          c=i;
512          p=node[c].parent;
513          while(p!=-1){
514              if(node[p].lchild==c)
515                  temp[start]=0;
516              else
517                  temp[start]=1;
518              start--;
519              c=p;
520              p=node[c].parent;
521          }
522          index=0;
523          for(j=start+1;j<amount;j++){
524              book[i].bit[index]=temp[j];
525              // printf("%d",temp[j]);
526              index++;
527          }
528          book[i].num=index;
529      }
530
531      // outout book
532      for(i=0;i<amount;i++){
533          fprintf(fp,"%d  ",node[i].symbol);
534          for(j=0;j<book[i].num;j++){
535              fprintf(fp,"%d",book[i].bit[j]);
536          }
537          fprintf(fp,"\n");
538      }
539      fclose(fp);
540  }
```

DE-ZIGZAG & INVERSE DPCM

```
388  YCbCr** dezigzag(int Nh,int Nw,int *AC_Y,int *AC_Cb,int *AC_Cr,int *DC_Y,int *DC_Cb,int *DC_Cr){
389      int m,n,i,j,index=0;
390      YCbCr **output=YCbCr_2D(Nh,Nw);
391       for(m=0;m<Nh;m+=8){
392           for(n=0;n<Nw;n+=8){
393               for(i=1;i<64;i++){
394                   output[m+row[i]][n+col[i]].Y = AC_Y[index];
395                   output[m+row[i]][n+col[i]].Cb = AC_Cb[index];
396                   output[m+row[i]][n+col[i]].Cr = AC_Cr[index];
397                   index++;
398               }
399           }
400       }
```

```
401       // put in DC
402       index=0;
403       for(i=0;i<Nh/8;i++){
404           for(j=0;j<Nw/8;j++){
405               output[i*8][j*8].Y = DC_Y[index];
406               output[i*8][j*8].Cb = DC_Cb[index];
407               output[i*8][j*8].Cr = DC_Cr[index];
408               index++;
409           }
410       }
411       // do in first column (DC)
412       for(i=8;i<Nh;i+=8){
413           output[i][0].Y += output[i-8][0].Y;
414           output[i][0].Cb += output[i-8][0].Cb;
415           output[i][0].Cr += output[i-8][0].Cr;
416       }
417       // do in every row (DC)
418       for(i=0;i<Nh;i+=8){
419           for(j=8;j<Nw;j+=8){
420               output[i][j].Y += output[i][j-8].Y;
421               output[i][j].Cb += output[i][j-8].Cb;
422               output[i][j].Cr += output[i][j-8].Cr;
423           }
424       }
425       return output;
426   }
```

IRLE

```
459  int* IRLE(int *input,int rle_size,int size){
460      int i,j=0,z,index=0,*output=calloc(size,sizeof(int));
461      for(i=1;i<=size/63;i++){
462          while(input[j]!=0||input[j+1]!=0){
463              for(z=0;z<input[j];z++){
464                  output[index]=0;
465                  index++;
466              }
467              output[index]=input[j+1];
468              index++;
469              j+=2;
470          }
471          while(index<(i*63)){
472              output[index]=0;
473              index++;
474          }
475          j+=2;
476      }
477      return output;
478  }
```

DEQUATIZATION

```
7   // quantization table (luminance)
8   const int Q_lumi[8][8]={
9       16,  11,  10,  16,  24,  40,  51,  61,
10      12,  12,  14,  19,  26,  58,  60,  55,
11      14,  13,  16,  24,  40,  57,  69,  56,
12      14,  17,  22,  29,  51,  87,  80,  62,
13      18,  22,  37,  56,  68, 109, 103,  77,
14      24,  35,  55,  64,  81, 104, 113,  92,
15      49,  64,  78,  87, 103, 121, 120, 101,
16      72,  92,  95,  98, 112, 100, 103,  99,
17  };
```

```
// quantization table (chorminance)
const int Q_chorm[8][8]={
    17,  18,  24,  47,  99,  99,  99,  99,
    18,  21,  26,  66,  99,  99,  99,  99,
    24,  26,  56,  99,  99,  99,  99,  99,
    47,  66,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
    99,  99,  99,  99,  99,  99,  99,  99,
};
```

IDCT

```
243  YCbCr** IDCT(YCbCr **input, int Nh, int Nw, double **cosi){
244      YCbCr **output = YCbCr_2D(Nh, Nw);
245      int m,n,u,v,x,y;
246      double *z,c;
247      z=calloc(3, sizeof(double));
248      for(m=0;m<Nh;m+=8){
249          for(n=0;n<Nw;n+=8){
250              for(x=0;x<8;x++){
251                  for(y=0;y<8;y++){
252                      z[0]=0.0;
253                      z[1]=0.0;
254                      z[2]=0.0;
255                      for(u=0;u<8;u++){
256                          for(v=0;v<8;v++){
257                              c=cosi[x][u] * cosi[y][v];
258                              if(u==0) c/=sqrt(2);
259                              if(v==0) c/=sqrt(2);
260                              z[0]+=input[m+u][n+v].Y * c;
261                              z[1]+=input[m+u][n+v].Cb * c;
262                              z[2]+=input[m+u][n+v].Cr * c;
263                          }
264                      }
265                      output[m+x][n+y].Y = z[0]/4+128;
266                      output[m+x][n+y].Cb = z[1]/4;
267                      output[m+x][n+y].Cr = z[2]/4;
268                  }
269              }
270
271          }
272      }
273      free(z);
274      return output;
275  }
```

RGB TO YCBCR

$$Y=0.299*R + 0.587*G + 0.114*B$$

$$Cb=0.564*(B-Y)$$

$$Cr=0.713*(R-Y)$$

YCBCR TO RGB

$$R=Y + 1.402*Cr;$$

$$G=Y - 0.344*Cb - 0.714*Cr$$

$$B=Y + 1.722*Cb$$

THANKS !