**实验三 数字图像编码实验**

**班级：**

**学号：**

**姓名：**

**一、实验目的**

1、理解有损压缩和无损压缩的概念；

2、理解图像压缩的主要原则和目的；

3、了解几种常用的图像压缩编码方式；

4、利用MATLAB程序进行图像压缩编码。

**二、实验原理**

图像压缩主要目的是为了节省存储空间，增加传输速度。图像压缩的理想标准是信息丢失最少，压缩比例最大。不损失图像质量的压缩称为无损压缩，无损压缩不可能达到很高的压缩比；损失图像质量的压缩称为有损压缩，高的压缩比是以牺牲图像质量为代价的。压缩的实现方法是对图像重新进行编码，希望用更少的数据表示图像。

信息的冗余量有许多种，如空间冗余，时间冗余，结构冗余，知识冗余，视觉冗余等，数据压缩实质上是减少这些冗余量。高效编码的主要方法是尽可能去除图像中的冗余成分，从而以最小的码元包含最大的图像信息。

编码压缩方法有许多种，从不同的角度出发有不同的分类方法，从信息论角度出发可分为有损和无损两大类，无损压缩编码如：哈夫曼（Huffman）编码，算术编码，行程（RLE）编码等。有损压缩编码包括预测编码，变换编码等。

本实验主要利用MATLAB程序进行哈夫曼编码、行程编码，并了解JPEG编码基本概念。

**三、实验内容**

1打开计算机，启动MATLAB程序；

2打开多幅图像(>3)，利用参考程序进行哈夫曼和行程编码压缩处理；计算原图和压缩以后的尺寸，计算压缩率并比较分析；

3查阅JPEG编码的有关资料，使用imwrite函数对图像进行JPEG压缩，质量因子分别选为20，60，80，对比显示原图与不同质量因子下解码后的图像，并显示均方根误差、压缩比、图像大小；

4记录和整理实验报告，总结实验心得。

哈夫曼参考程序：

function Huffman()

clear

close all;

X=imread('girl.jpg');

data=uint8(X);

[zipped,info]=huffencode(data);

unzipped=huffdecode(zipped,info);

subplot(121);imshow(data);

subplot(122);imshow(unzipped);

erms=0

cr=info.ratio

whos data unzipped zipped

function [zipped,info]=huffencode(vector)

if ~isa(vector,'uint8')

error('input argument must be a uint8 vector');

end

[m,n]=size(vector);

vector=vector(:)';

f=frequency(vector);

simbols=find(f~=0);

f=f(simbols);

[f,sortindex]=sort(f);

simbols=simbols(sortindex);

len=length(simbols);

simbols\_index=num2cell(1:len);

codeword\_tmp=cell(len,1);

while length(f)>1

index1=simbols\_index{1};

index2=simbols\_index{2};

codeword\_tmp(index1)=addnode(codeword\_tmp(index1),uint8(0));

codeword\_tmp(index2)=addnode(codeword\_tmp(index2),uint8(1));

f=[sum(f(1:2)) f(3:end)];

simbols\_index=[{[index1,index2]} simbols\_index(3:end)];

[f,sortindex]=sort(f);

simbols\_index=simbols\_index(sortindex);

end

codeword=cell(256,1);

codeword(simbols)=codeword\_tmp;

len=0;

for index=1:length(vector)

len=len+length(codeword{double(vector(index))+1});

end

string=repmat(uint8(0),1,len);

pointer=1;

for index=1:length(vector)

code=codeword{double(vector(index))+1};

len=length(code);

string(pointer+(0:len-1))=code;

pointer=pointer+len;

end

len=length(string);

pad=8-mod(len,8);

if pad>0

string=[string uint8(zeros(1,pad))];

end

codeword=codeword(simbols);

codelen=zeros(size(codeword));

weights=2.^(0:23);

maxcodelen=0;

for index=1:length(codeword)

len=length(codeword{index});

if len>maxcodelen

maxcodelen=len;

end

if len>0

code=sum(weights(codeword{index}==1));

code=bitset(code,len+1);

codeword{index}=code;

codelen(index)=len;

end

end

codeword=[codeword{:}];

cols=length(string)/8;

string=reshape(string,8,cols);

weights=2.^(0:7);

zipped=uint8(weights\*double(string));

huffcodes=sparse(1,1);

for index=1:nnz(codeword)

huffcodes(codeword(index),1)=simbols(index);

end

info.pad=pad;

info.huffcodes=huffcodes;

info.ratio=cols./length(vector);

info.length=length(vector);

info.maxcodelen=maxcodelen;

info.rows=m;

info.cols=n;

function vector=huffdecode(zipped,info,image)

if ~isa(zipped,'uint8')

error('input argument must be a uint8 vector');

end

len=length(zipped);

string=repmat(uint8(0),1,len.\*8);

bitindex=1:8;

for index=1:len

string(bitindex+8.\*(index-1))=uint8(bitget(zipped(index),bitindex));

end

string=logical(string(:)');

len=length(string);

string((len-info.pad+1):end)=[];

len=length(string);

weights=2.^(0:51);

vector=repmat(uint8(0),1,info.length);

vectorindex=1;

codeindex=1;

code=0;

for index=1:len

code=bitset(code,codeindex,string(index));

codeindex=codeindex+1;

byte=decode(bitset(code,codeindex),info);

if byte>0

vector(vectorindex)=byte-1;

codeindex=1;

code=0;

vectorindex=vectorindex+1;

end

end

vector=reshape(vector,info.rows,info.cols);

function codeword\_new=addnode(codeword\_old,item)

codeword\_new=cell(size(codeword\_old));

for index=1:length(codeword\_old)

codeword\_new{index}=[item codeword\_old{index}];

end

function f=frequency(vector)

if ~isa(vector,'uint8')

error('input argument must be a uint8 vector');

end

f=repmat(0,1,256);

len=length(vector);

for index=0:255

f(index+1)=sum(vector==uint8(index));

end

f=f./len;

function byte=decode(code,info)

byte=info.huffcodes(code); %%程序结束

行程编码参考程序：

function RLE()

clear;

close all;

I = imread('girl.jpg');

[r\_ c\_ d\_] = size(I);

if d\_ > 1

I0=rgb2gray(I);

else

I0=I;

end

I0=round(I0/20)\*20;

[zipped,info]=RLEncode(I0);%%调用RLE进行编码

unzipped=RLEdecode(zipped,info); %%调用解码程序进行编码

subplot(121);imshow(I0);

subplot(122);imshow(unzipped);

%erms=compare(I(:),unzipped(:))

cr=info.ratio

whos I unzipped zipped

function[zipped,info]=RLEncode(vector)

[m,n]=size(vector);

vector=vector(:)';

L=length(vector);

c=vector(1);e(1,1)=c;e(1,2)=0;%e(:,1)存放灰度，e(:,2)存放行程

t1=1

for j=1:L

if((vector(j)==c))

e(t1,2)=double(e(t1,2))+1;

else

c=vector(j);

t1=t1+1;

e(t1,1)=c;

e(t1,2)=1;

end

end

zipped=e;

info.rows=m;

info.cols=n;

[m,n]=size(e);

info.ratio=m\*n/(info.rows\*info.cols);

function unzipped=RLEdecode(zip,info)

zip=uint8(zip);

[m,n]=size(zip);

unzipped=[];

for i=1:m

section=repmat(zip(i,1),1,double(zip(i,2)));

unzipped=[unzipped section];

end

unzipped=reshape(unzipped,info.rows,info.cols);%程序结束

%JPEG压缩参考程序

clear;

close all;

factor = 50

x=imread('lena.png');

imwrite(x,'lena1.jpg','quality', factor); %根据题目要求修改质量因子

x1=imread('lena1.jpg');

k1=imfinfo('lena1.jpg')

s1=k1.FileSize/1024;

e1=x(:)-x1(:); %减去原图得到误差

[m1,n1]=size(e1);

erms1=sqrt(sum(e1(:)).^2/(m1\*n1));%求均方根误差

i\_size1=k1.Width\*k1.Height\*k1.BitDepth/8;

i\_compress1=k1.FileSize;

ratio1=i\_size1/i\_compress1;

subplot(221);

imshow(x1);

fprintf('质量因子:%d\n',factor);

fprintf('均方根误差:%f\n',erms1);

fprintf('压缩比：%f\n',ratio1);

fprintf('压缩后图像大小：%f\n',i\_compress1);