REFERENCES

1. Http://www.swin.edu.au/astronomy/pbourke/colour/colourspace/
2. Http://www.swin.edu.au/astronomy/pbourke/indexcolour/
3. Baxes, G.A. *Digital image processing: principles and applications.* New York: John Wiley & Sons, 1994, p 179.
4. Weeks, A.R., Jr. *Fundamental of Electronic Image Processing.* Washington: SPIE Press, 1996, p 472.
5. Weeks, A.R., Jr. *Fundamental of Electronic Image Processing.* Washington: SPIE Press, 1996, p 482.
6. Baxes, G.A. *Digital image processing: principles and applications.* New York: John Wiley & Sons, 1994, p 181.
7. Weeks, A.R., Jr. *Fundamental of Electronic Image Processing.* Washington: SPIE Press, 1996, p 473.
8. Gonzalez, R.C. and Woods, R.E. *Digital image processing.* Addison-Wesley, 1992, p 343.
9. Weeks, A.R., Jr. *Fundamental of Electronic Image Processing.* Washington: SPIE Press, 1996, p 475.
10. Weeks, A.R., Jr. *Fundamental of Electronic Image Processing.* Washington: SPIE Press, 1996, p 477.
11. Gonzalez, R.C. and Woods, R.E. *Digital image processing.* Addison-Wesley, 1992, p 317.
12. Rabbani, M. and Jones, P.W. *Digital Image Compression Techniques.* Washington: SPIE Press, 1991, p 6.

REFERENCES (continued)

13. Baxes, G.A. *Digital image processing: principles and applications*. New York: John Wiley & Sons, 1994, p 192.

14. Gonzalez, R.C. and Woods, R.E. *Digital image processing*. Addison-Wesley, 1992, p 343.

15. Dougherty, E.R. *Digital Image Processing Methods*. New York: Maral Dekker, 1994, p 268.

16. Gormish, M. J. Compression of Palettized Images by Color. *Proceeding of the IEEE International Conference on Image Processing* (1995) : 274-277.

17. Jaisimha, M. Y., Potlapalli, H., Barad, H., Martinez, A. B., Lohrenz, M. C., Ryan, J., and Pollard, J. Data Compression Techniques For Maps. *Proceeding of the IEEE Southeast Congress on Energy and Information Technologies* (1989) : 878-883.

18. Liu, N., Yan, H., and Martin, P. Segmentation of Map Image Using Opponent Color Dimensions. *J. Color Research and Application.* **21(2)**(1996) : 115-120.

19. Ou, Y., Huang, J., and Yu, Y. A Practical Data Compression System for Map Images. *Proceeding of IEEE TENCON. on Digital Signal Processing Applications* (1996): 698-701.

20. Overloop, J. V., Philips, W., Torfs, D., and Lemahieu, I. Segmented Image Coding of Palettized Images. *Procedding of IEEE International Conference on The Acoustics, Speech, and Signal Processing* (1997): 2937-2939.

REFERENCES (continued)

21. Weeks, A.R., Jr. *Fundamental of Electronic Image Processing*. Washington: SPIE Press, 1996, p 501.

22. Murray, J. D. and Ryper, W. v. *Encyclopedia of Graphics File Formats*. CA: O'Reilly & Associates, 1994, p 132.

23. Murray, J. D. and Ryper, W. v. *Encyclopedia of Graphics File Formats*. CA: O'Reilly & Associates, 1994, p 138.

APPENDICES

APPENDIX A

RTSDZIP PROGRAM WINDOWS

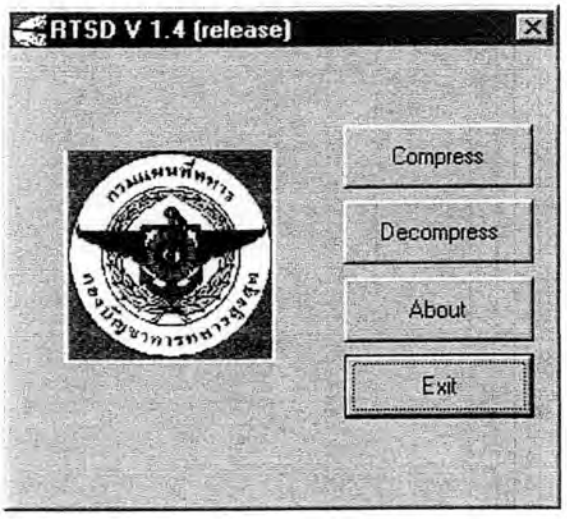

Figure A-1 : RTSDZIP main window

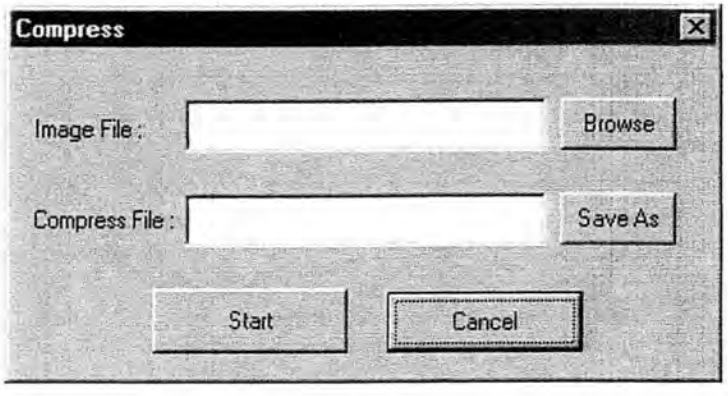

Figure A-2 : RTSDZIP compression window

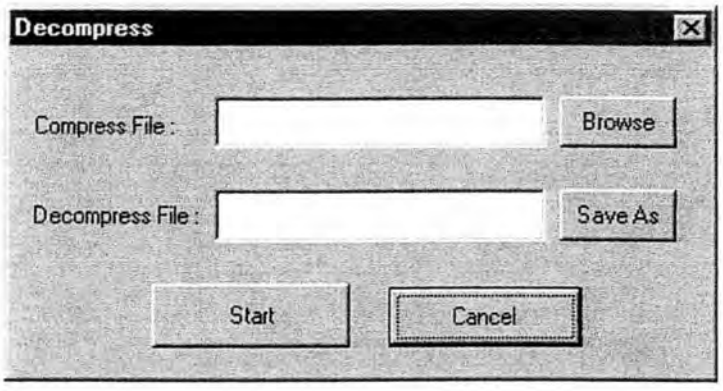

Figure A-3 : RTSDZIP decompression window

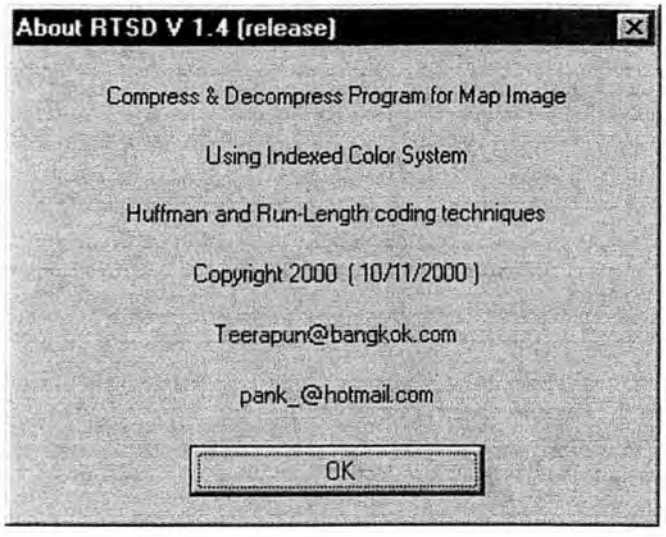

Figure A-4 : RTSDZIP about window

APPENDIX B

RTSDZIP Compression-Decompression Algorithms Source Code

```
#ifndef _HMAN_H
#define _HMAN_H

//linklist
class node
{
public:
        node *next;
        BYTE r;
        BYTE g;
        BYTE b;
        BYTE index;
        node()
        {
                next=NULL;
                r=0;
                g=0;
                b=0;
                index=0;
        }
        ~node()
        {
        }
        void operator = (const node &in)
        {
                r = in.r;
                g = in.g;
                b = in.b;
                next = in.next;
```

```
            index = in.index;
        }
        int operator == (const node &in)const
        {
            if(r==in.r && g==in.g && b==in.b) return 1;
            return 0;
        }
};
void append_node(node *&root,const BYTE &r,const BYTE &g,const BYTE
&b,const BYTE &index)
{
        if(root==NULL)
        {
            root = new node;
            root->r = r;
            root->g = g;
            root->b = b;
            root->index = index;
        }else{
            node *tmp = new node;
            tmp->r = r;
            tmp->g = g;
            tmp->b = b;
            tmp->next = root;
            tmp->index = index;
            root = tmp;
        }
}
void delete_node(node *&root)
{
        node *root1;
        while(root)
        {
            root1 = root->next;
```

```
                delete root;
                root = root1;
        }
}

class bucket
{
public:
        node *root;
        bucket()
        {
                root=NULL;
        }
        ~bucket()
        {
                delete_node(root);
        }
        void append(const BYTE &r,const BYTE &g,const BYTE &b,const BYTE
&index)
        {
                append_node(root,r,g,b,index);
        }
        node *find(const BYTE &r,const BYTE &g,const BYTE &b)
        {
                node *root1 = root;
                while(root1){
                        if(root1->r==r && root1->g==g && root1->b==b) break;
                        root1 = root1->next ;
                }
                return root1;
        }
};
//hash table
class hash
```

```
{
public:
        bucket list[997];
        hash()
        {
        }
        ~hash()
        {
        }
        node * has(const BYTE &r,const BYTE &g,const BYTE &b)
        {
                int tmp = RGB(r,g,b)%997;
                node *troot = list[tmp].find(r,g,b);
                return troot;
        }

        node * append(const BYTE &r,const BYTE &g,const BYTE &b,const BYTE
&index)
        {
                int tmp = RGB(r,g,b)%997;
                node *troot = list[tmp].find(r,g,b);
                if(troot)
                {
                        return troot;
                }
                list[tmp].append(r,g,b,index);
                return list[tmp].root;
        }
};
//file
class T_file
{
        HANDLE fp;
public:
```

```
int flag;//1="r"(read) ; 2="w"(write)
T_file()
{
        fp=NULL;
        flag=0;
}
T_file(const char *fname,const char *mode)
{
        flag=0;
        fp==NULL;
        Open(fname,mode);
}
~T_file()
{
        Close();
}
int Open(const char *fname,const char *mode)
{
        Close();
        if(mode[0]=='r' || mode[0]=='R') flag=1;
        else if(mode[0]=='w' || mode[0]=='W') flag=2;
        if(flag==0) return 0;
        if(flag==1)

    fp=CreateFile(fname,GENERIC_READ,/*0*/FILE_SHARE_READ,NULL,O
PEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL);
        else if(flag==2){
    fp=CreateFile(fname,GENERIC_WRITE,0/*FILE_SHARE_WRITE*/,NULL
,CREATE_NEW,FILE_ATTRIBUTE_NORMAL,NULL);
                if(fp==INVALID_HANDLE_VALUE)
    fp=CreateFile(fname,GENERIC_WRITE,0/*FILE_SHARE_WRITE*/,NULL
,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
        }
        if(fp==INVALID_HANDLE_VALUE){fp=NULL; flag=0; return 0;}
```

```
        return 1;
}
void Close()
{
        if(flag)
                CloseHandle(fp);
        flag=0;
}
int Length()const
{
        if(flag==0) return 0;
        int len=0;
        return GetFileSize(fp,(DWORD *)&len);
}
int Read(void *ptr,int size,int len)
{
        if(flag==0) return 0;
        unsigned long rsize;
        ReadFile(fp,ptr,size*len,&rsize,NULL);
        return rsize;
}
int Write(void *ptr,int size,int len)
{
        if(flag==0) return 0;
        unsigned long rsize;
        WriteFile(fp,ptr,size*len,&rsize,NULL);
        return rsize;
}
int Seek(int offs,DWORD MoveMethod=FILE_CURRENT)
{
        if(flag==0) return 0;
        LARGE_INTEGER li;
        li.QuadPart = offs;
```

```
            li.LowPart = SetFilePointer (fp, li.LowPart, &li.HighPart,
MoveMethod);
            if (li.LowPart == 0xFFFFFFFF && GetLastError() != NO_ERROR)
            {
                li.QuadPart = -1;
            }
            return li.QuadPart;
        }
};
//compress by huffman + index color + runlength
int compress_color_index_runlength(char *file_name1,char *file_name2)
{
        int i,j,N,M,loc,k,ind,xt,yt;
        unsigned char *v,*L,*gray,*table_length;
        unsigned long int *code,*table_code,ctemp2;
        unsigned long int aux1,aux2,Lmask,act_len,flength;
        unsigned char mask,Len;
        int ch;
        unsigned char ctemp,Ltemp;
        float temp,sum,*pt,*p,big;
        unsigned long int *histo;
        double nsq;
        FILE *fptr,*fptro;
        fptro=fopen(file_name2,"wb");
        if(fptro==NULL)
        {
            return 0;
        }
        /*Generate histogram*/
        histo=(unsigned long int *)malloc(256*sizeof(long int));
        memset(histo,0,256*sizeof(long int));
//index color
        T_file fp11,fp12,fp13,fp2;
        if(!fp11.Open(file_name1,"r")) return 0;
```

```
        if(!fp12.Open(file_name1,"r")) return 0;
        if(!fp13.Open(file_name1,"r")) return 0;
        if(!fp2.Open("_##tmp_","w")) return 0;
        int tlen = fp11.Length()/3;
        BYTE ir,ig,ib,ilen=3;
        BYTE tr[256],tg[256],tb[256];
        memset(tr,0,256);
        memset(tg,0,256);
        memset(tb,0,256);
        fp2.Write(tr,1,256);
        fp2.Write(tg,1,256);
        fp2.Write(tb,1,256);
        hash htable;
//runlength
        BYTE prev;
        int plen=1;
        fp12.Seek(tlen);
        fp13.Seek(2*tlen);

        for(int iq=0;iq<tlen;iq++)
        {
                fp11.Read(&ir,1,1);
                fp12.Read(&ig,1,1);
                fp13.Read(&ib,1,1);
                node *tmp_node = htable.has(ir,ig,ib);
                if(tmp_node){
                        if(tmp_node->index==prev){
                                plen++;  //runlength
                        }else{
                                if(plen==1){
                                        fp2.Write(&(tmp_node->index),1,1);
                                        histo[tmp_node->index]++;
                                        prev = tmp_node->index;
                                        plen=1;
```

```
}else if(plen<3){
        fp2.Write(&prev,1,1);
        histo[prev]++;
        fp2.Write(&(tmp_node->index),1,1);
        histo[tmp_node->index]++;
        prev = tmp_node->index;
        plen=1;
}else{ //>=3
        if(plen<256){
                BYTE tt = 0;
                fp2.Write(&tt,1,1);
                histo[tt]++;
                tt = plen;
                fp2.Write(&tt,1,1);
                histo[tt]++;
        }else if(plen<65536){
                BYTE tt = 1;
                fp2.Write(&tt,1,1);
                histo[tt]++;
                unsigned short tt1 = plen;
                fp2.Write(&tt1,sizeof(unsigned short),1);
                BYTE tt2[2];
                memcpy(tt2,&tt1,2);
                histo[tt2[0]]++;
                histo[tt2[1]]++;
        }else{
                BYTE tt = 2;
                fp2.Write(&tt,1,1);
                histo[tt]++;
                unsigned int tt1 = plen;
                fp2.Write(&tt1,sizeof(int),1);
                BYTE tt2[4];
                memcpy(tt2,&tt1,4);
                histo[tt2[0]]++;
```

```
                        histo[tt2[1]]++;
                        histo[tt2[2]]++;
                        histo[tt2[3]]++;
                    }
                    fp2.Write(&(tmp_node->index),1,1);
                    histo[tmp_node->index]++;
                    prev = tmp_node->index;
                    plen=1;
                }
            }
        }else{
            if(plen==1){
            }else if(plen<3){
                fp2.Write(&prev,1,1);
                histo[prev]++;
            }else{ //>=3
                if(plen<256){
                    BYTE tt = 0;
                    fp2.Write(&tt,1,1);
                    histo[tt]++;
                    tt = plen;
                    fp2.Write(&tt,1,1);
                    histo[tt]++;
                }else if(plen<65536){
                    BYTE tt = 1;
                    fp2.Write(&tt,1,1);
                    histo[tt]++;
                    unsigned short tt1 = plen;
                    fp2.Write(&tt1,sizeof(unsigned short),1);
                    BYTE tt2[2];
                    memcpy(tt2,&tt1,2);
                    histo[tt2[0]]++;
                    histo[tt2[1]]++;
                }else{
```

```
                                BYTE tt = 2;
                                fp2.Write(&tt,1,1);
                                histo[tt]++;
                                unsigned int tt1 = plen;
                                fp2.Write(&tt1,sizeof(int),1);
                                BYTE tt2[4];
                                memcpy(tt2,&tt1,4);
                                histo[tt2[0]]++;
                                histo[tt2[1]]++;
                                histo[tt2[2]]++;
                                histo[tt2[3]]++;
                        }
                }
                htable.append(ir,ig,ib,ilen);
                fp2.Write(&ilen,1,1);
                tr[ilen] = ir;
                tg[ilen] = ig;
                tb[ilen] = ib;
                prev = ilen;
                plen = 1;
                histo[ilen]++;
                ilen++;

        if(ilen==253){
                fcloseall();
                fp2.Close();
                free(histo);
                DeleteFile("_##tmp_");
                ::MessageBox(NULL, "!More index > 253
color","Error",MB_OK);
                return 0;
        }
        }
    }
```

```
//runlength
if(plen==1){
}else if(plen<3){
        fp2.Write(&prev,1,1);
        histo[prev]++;
}else{ //>=3
        if(plen<256){
                BYTE tt = 0;
                fp2.Write(&tt,1,1);
                histo[tt]++;
                tt = plen;
                fp2.Write(&tt,1,1);
                histo[tt]++;
        }else if(plen<65536){
                BYTE tt = 1;
                fp2.Write(&tt,1,1);
                histo[tt]++;
                unsigned short tt1 = plen;
                fp2.Write(&tt1,sizeof(unsigned short),1);
                BYTE tt2[2];
                memcpy(tt2,&tt1,2);
                histo[tt2[0]]++;
                histo[tt2[1]]++;
        }else{
                BYTE tt = 2;
                fp2.Write(&tt,1,1);
                histo[tt]++;
                unsigned int tt1 = plen;
                fp2.Write(&tt1,sizeof(int),1);
                BYTE tt2[4];
                memcpy(tt2,&tt1,4);
                histo[tt2[0]]++;
                histo[tt2[1]]++;
                histo[tt2[2]]++;
```

```
                histo[tt2[3]]++;
        }
}
fp11.Close();
fp12.Close();
fp13.Close();
fp2.Seek(0,FILE_BEGIN);
fp2.Write(tr,1,256);
fp2.Write(tg,1,256);
fp2.Write(tb,1,256);
fp2.Close();
for(iq=0;iq<256;iq++){
        histo[tr[iq]]++;
        histo[tg[iq]]++;
        histo[tb[iq]]++;
}
fptr=fopen("_##tmp_","rb");
if(fptr==NULL)
{
        DeleteFile("_##tmp_");
        return 0;
}
p=(float *)malloc(256*sizeof(float));
gray=(unsigned char *)malloc(256*sizeof(char));
k=0;
for(i=0;i<256;i++)
{
        if(histo[i]!=0)
        {
                p[k]=(float)histo[i];
                gray[k]=i;
                k++;
        }
}
```

```
free(histo);
N=k;
/*Normalize*/
sum=0.0;
for(i=0;i<N;i++)
sum+=p[i];
for(i=0;i<N;i++)
p[i]/=sum;
/*Rearanging the probability values in ascending order*/
for(i=0;i<(N-1);i++)
{
        big=p[i];
        loc=i;
        for(j=i+1;j<N;j++)
        {
                if(p[j]>big)
                {
                        big=p[j];
                        loc=j;
                }
        }
        if(loc==i) continue;
        else
        {
                temp=p[i];
                ctemp=gray[i];
                p[i]=p[loc];
                gray[i]=gray[loc];
                p[loc]=temp;
                gray[loc]=ctemp;
        }
}
pt=(float *)malloc(N*sizeof(float));
memcpy(pt,p,N*sizeof(float));
```

```
v=(unsigned char *)malloc((N-2)*sizeof(char));
code=(unsigned long int *)malloc(N*sizeof(unsigned long int));
L=(unsigned char *)malloc(N*sizeof(char));

memset(v,0,N-2);
/*Contraction steps in the generation of the Huffman's code*/
M=N;
for(i=0;i<(N-2);i++)
{
        p[M-2]=p[M-1]+p[M-2];
        loc=M-2;
        for(j=0;j<(M-1);j++)
        {
                if(p[M-2]>=p[j])
                {
                        loc=j;
                        break;
                }
        }
        temp=p[M-2];
        for(j=M-2;j>=loc;j--)
                p[j]=p[j-1];
        p[loc]=temp;
        M--;
        v[(N-3)-i]=loc;
}
/*Expansion steps in the generation of the Huffman's codes*/
memset(code,0,N*sizeof(unsigned long int));
memset(L,1,N);

code[0]=0;
code[1]=1;
M=1;
```

```
for(i=0;i<(N-2);i++)
{
        if(v[i]==M)
        {
                code[M+1]=(code[M]<<1)+1;
                code[M]=(code[M])<<1;
                L[M]++;
                L[M+1]=L[M];
        }
        else
        {
                ctemp2=code[v[i]];
                Ltemp=L[v[i]];
                for(j=v[i];j<M;j++)
                {
                        code[j]=code[j+1];
                        L[j]=L[j+1];
                }
                code[M]=ctemp2;
                L[M]=Ltemp;
                code[M+1]=(code[M]<<1)+1;
                code[M]=code[M]<<1;
                L[M]++;
                L[M+1]=L[M];
        }
        M++;
}
free(v);
free(p);
free(pt);

/*Coding*/
/*Writing header of the output file.
The first 4 bytes stores the true length in
```

```
bits of the stored image with the MSB stored
first. The 5th byte is the number of Huffman
codes=N. The following N bytes contain the N
natural codes for the gray levels, followed by N
bytes containing the Huffman code lengths. These
are then followed by the actual Huffman codes
stored in packed form.*/

for(i=0;i<4;i++)
putc((int)0,fptro); /*reserve the first 4 bytes for*/
putc(N,fptro);     /*the true length of the file in bits*/
fwrite(gray,1,N,fptro);
fwrite(L,1,N,fptro);
aux1=0;
Len=0;
for(i=0;i<N;i++)
{
        Len+=L[i];
        aux1=(aux1<<L[i])|code[i];
        if(Len<8) continue;
        else
                while(Len>8)
                {
                        aux2=aux1;
                        Lmask=255;
                        Lmask<<=(Len-8);
                        aux2=(aux2&Lmask);
                        aux2>>=(Len-8);
                        ch=(char)aux2;
                        putc(ch,fptro);
                        Lmask=~Lmask;
                        aux1=aux1&Lmask;
                        Len-=8;
                }
```

```
}
if(aux1!=0)
{
        ch=(char)(aux1<<(8-Len));
        putc(ch,fptro);
}
table_code=(unsigned long int *)malloc(256*sizeof(long int));
table_length=(unsigned char *)malloc(256*sizeof(char));
for(i=0;i<N;i++)
{
        table_code[gray[i]]=code[i];
        table_length[gray[i]]=L[i];
}
rewind(fptr);
Len=0;
act_len=0;/*variable to save true length of file in bits*/
aux1=aux2=0;
k=0;
while((ch=(unsigned int)getc(fptr))!=EOF)
{
        k++;
        Len+=table_length[ch];
        act_len+=table_length[ch];
        aux1=(aux1<<table_length[ch])|table_code[ch];
        if(Len<8) continue;
        else
                while(Len>=8)
                {
                        aux2=aux1;
                        Lmask=255;
                        Lmask<<=(Len-8);
                        aux2=aux2&Lmask;
                        Lmask=~Lmask;
                        aux1=aux1&Lmask;
```

```
                        aux2>>=(Len-8);
                        ch=(char)aux2;
                        Len-=8;
                        putc(ch,fptro);
                }
        }
        if(aux1!=0 || Len!=0)/*transmit remain bits*/
        {
                ch=(char)(aux1<<(8-Len));
                putc(ch,fptro);
        }
        rewind(fptro);
        /*Write true length of file*/
        M=8*sizeof(long int)-8;
        for(i=0;i<4;i++)
        {
                Lmask=255;
                Lmask<<=M;
                aux1=(act_len&Lmask);
                aux1>>=M;
                ch=(int)aux1;
                putc(ch,fptro);
                M-=8;
        }
        fclose(fptro);
        fclose(fptr);

        DeleteFile("_##tmp_");
        return 1;
}
//decompress
int decompress_color_index_runlength(char *file_name1,char *file_name2)
{
        unsigned int i,j,N,M,k,xt,yt;
```

```
unsigned char *L,*gray;
unsigned long int *code;
unsigned long int aux1,aux2,Lmask,act_len,flength;
unsigned char mask,Len;
int ch,ind;
unsigned char ctemp,Ltemp;
FILE *fptr;

fptr=fopen(file_name1,"rb");
if(fptr==NULL)
{
        return 0;
}
/*Reading the header*/
act_len=0;
M=8*sizeof(long int)-8;
for(i=0;i<4;i++)
{
        ch=(unsigned int)getc(fptr);
        aux1=(unsigned long int)ch;
        aux1<<=M;
        act_len|=aux1;
        aux1=(long int)0;
        M-=8;
}
N=getc(fptr);
if(N==0) N=256;
gray=(unsigned char *)malloc(N*sizeof(char));
L=(unsigned char *)malloc(N*sizeof(char));
code=(unsigned long int *)malloc(N*sizeof(long int));

fread(gray,1,N,fptr);

fread(L,1,N,fptr);
```

```
mask=128;
Len=0;
aux2=0;
aux1=0;
ind=1;
i=0;
while(ind)
{
        ch=(unsigned int)getc(fptr);
        for(k=0;k<8;k++)
        {
                aux1=ch&mask;
                ch<<=1;
                aux2<<=1;
                if(aux1!=0)
                        aux2|=1;
                Len++;
                if(Len==L[i])
                {
                        code[i]=aux2;
                        aux2=0;
                        Len=0;
                        i++;
                        if(i==N)
                        {
                                ind=0;
                                break;
                        }
                }
        }
}
Len=0;
aux1=aux2=0;
flength=0; /*a parameter measure against actual file length*/
```

```
        mask=128;
        ind=1;
//index color
        T_file fp1(file_name2,"w");
        T_file fp2("_##tmp2_","w");
        T_file fp3("_##tmp3_","w");
        BYTE ir[256],ig[256],ib[256];
        int ilen=0,it=0;
//runlength
        BYTE prev,ibyte[4],icount=0,itype=3;
        while(ind)
        {
                ch=(unsigned int)getc(fptr);
                for(k=0;k<8;k++)
                { /*test if the actual end of the file has been reached*/
                        if(flength>=act_len)
                        {
                                ind=0;
                                break;
                        }
                        aux1=(ch&mask);
                        ch<<=1;
                        aux2<<=1;
                        if(aux1!=0)
                                aux2|=1;
                        Len++;
                        for(j=0;j<N;j++)
                        {
                                if(L[j]>Len)
                                {
                                        break;
                                }
                                if(L[j]==Len)
                                {
```

```
if(code[j]==aux2)
{
        aux2=0;
        if(ilen<256){
                if(it==0){
                        ir[ilen]=gray[j];
                }else if(it==1){
                        ig[ilen]=gray[j];
                }else{
                        ib[ilen]=gray[j];
                }
                if(ilen==255 && it<2){
                        ilen=0;
                        it++;
                }else
                        ilen++;
        }else{
                if(itype==3){

                        if(gray[j]==0){
                                itype=0;
                        }else if(gray[j]==1){
                                itype=1;
                        }else if(gray[j]==2){
                                itype=2;
                        }else{
                fp1.Write(&ir[gray[j]],1,1);
                fp2.Write(&ig[gray[j]],1,1);
                fp3.Write(&ib[gray[j]],1,1);
                                prev = gray[j];
                        }
                        goto done;
                }else if(itype==0){
```

```
BYTE *tmp_bit = new BYTE[gray[j]-1];
memset(tmp_bit,ir[prev],gray[j]-1);
fp1.Write(tmp_bit,1,gray[j]-1);
memset(tmp_bit,ig[prev],gray[j]-1);
fp2.Write(tmp_bit,1,gray[j]-1);
memset(tmp_bit,ib[prev],gray[j]-1);
fp3.Write(tmp_bit,1,gray[j]-1);
delete[] tmp_bit;
icount=0;
itype=3; //normal
}else if(itype==1){
ibyte[icount]=gray[j];
icount++;
if(icount==2){
unsigned short ll;
memcpy(&ll,ibyte,2);

BYTE *tmp_bit = new BYTE[ll-1];
memset(tmp_bit,ir[prev],ll-1);
fp1.Write(tmp_bit,1,ll-1);
memset(tmp_bit,ig[prev],ll-1);
fp2.Write(tmp_bit,1,ll-1);
memset(tmp_bit,ib[prev],ll-1);
fp3.Write(tmp_bit,1,ll-1);
delete[] tmp_bit;
icount=0;
itype=3; //normal
}
}else if(itype==2){
ibyte[icount]=gray[j];
icount++;
if(icount==4){
unsigned int ll,tmp_ll;
memcpy(&ll,ibyte,4);
```

```
BYTE *tmp_bit = new BYTE[1000000];
memset(tmp_bit,ir[prev],1000000);
tmp_ll = ll-1;
while(tmp_ll>1000000)
{
fp1.Write(tmp_bit,1,1000000);
tmp_ll-=1000000;
}
if(tmp_ll>0){
fp1.Write(tmp_bit,1,tmp_ll);
}
memset(tmp_bit,ig[prev],1000000);
tmp_ll = ll-1;
while(tmp_ll>1000000)
{
fp2.Write(tmp_bit,1,1000000);
tmp_ll-=1000000;
}
if(tmp_ll>0){
fp2.Write(tmp_bit,1,tmp_ll);
}
memset(tmp_bit,ib[prev],1000000);
tmp_ll = ll-1;
while(tmp_ll>1000000)
{
fp3.Write(tmp_bit,1,1000000);
tmp_ll-=1000000;
}
if(tmp_ll>0){
fp3.Write(tmp_bit,1,tmp_ll);
}

icount=0;
itype=3; //normal
```

```
                                        }
                                    }
                                }
done:
                                Len=0;
                                break;
                            }
                        }
                    }
                    flength++;
                }
            }

            fclose(fptr);
            fp2.Close();
            fp3.Close();
            BYTE bt;
            fp2.Open("_##tmp2_","r");
            while(fp2.Read(&bt,1,1))
                fp1.Write(&bt,1,1);
            fp2.Close();
            DeleteFile("_##tmp2_");
            fp3.Open("_##tmp3_","r");
            while(fp3.Read(&bt,1,1))
                fp1.Write(&bt,1,1);
            fp3.Close();
            DeleteFile("_##tmp3_");
            return 1;
}
#endif
```

APPENDIX C

OFFSET PRINTS USING THE ORIGINAL MAP IMAGE DATA AND THE OUTPUT FROM THE RTSDZIP

แทรก

แพนบก้!

APPENDIX D

DIGITAL PRINTS USING THE ORIGINAL MAP IMAGE DATA AND THE OUTPUT FROM THE RTSDZIP

แทรก

แผนที่ !

VITA

Lt.Teerapun Weladee was born on April 14, 1975 in Lampang, Thailand. He received his B.E. degree in Survey Engineering from Chulachomklao Royal Military Academy in 1998, and he has worked at the Royal Thai Survey Department. He has been a graduate student in Imaging Technology Program, Graduate School, Chulalongkorn University since 1999.