Subject: Re: COLOR_QUAN question
Posted by steinhh on Thu, 19 Aug 1999 07:00:00 GMT
View Forum Message <> Reply to Message

In article <37BC1A07.E6E4C022@ssec.wisc.edu>
Liam Gumley <Liam.Gumley@ssec.wisc.edu> writes:

> Have you looked at the ImageMagick API?
> http://www.wizards.dupont.com/cristy/www/api.html
>
> Seems like it could be interfaced to IDL by some enterprising
> individual.

Sure, no problem, I think. Other than how to find the Time...

Speaking of time, I just wrote a DLM wrapper for the one of the functions in the FFTW library (Fastest Fourier Transform in the West see http://www.fftw.org), to do real->complex and complex->real transforms of some *large* datasets. Beats the native IDL FFT by factors of about 6 - 15 on my platform! Of course it only uses half the space, as well.... Ideal for convolving fltarrs with other fltarrs!

In the spirit of sharing etc... here follows the DLM file and the c code to be linked with the fftw libraries. This one only deals with single precision transforms, mind you... Although the library can be configured to produce both double and single precision code, it's a bit difficult to mix transforms for different types in one executable...

And no, I won't have time to answer questions about how to get this working on your computer... But as a hint, my link statement includes "-lidl -lsrfftw -lsfftw -lc -lm".

Use: To convolve a=fltarr(m,n,...) with b=fltarr(m,n,...), do this:

```
result = rfftwnd(rfftwnd(a)*rfftwnd(b),(size(b))(1))
```

Bye for now.

Stein Vidar

rfftwnd.dlm---------- ---

```
# $Id: rfftwnd.dlm,v 1.1 1999/08/16 10:59:04 steinhh Exp steinhh $
MODULE RFFTWND
DESCRIPTION N-dimensional Real fftw
VERSION $Revision: 1.1 $
```

```
BUILD_DATE $Date: 1999/08/16 10:59:04 $
SOURCE S.V.H.HAUGAN
FUNCTION RFFTWND    1 2
 ------------------------------------------------------------ ---

 rfftwnd.c----------------------------------------------- ---
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "export.h"
#include "srfftw.h"

#define NULL_VPTR ((IDL_VPTR) NULL)

#define GETVARADDR(v) ((v->flags&IDL_V_ARR) ? (void*)v->value.arr->data \
    : (void*) &v->value.c)

#define GETVARDATA(v,n) ((v->flags&IDL_V_ARR) \
    ? (*(n) = v->value.arr->n_elts, v->value.arr->data) \
    : (*(n) = 1, & v->value.c ) )


FILE *rfftw_wisdom_file(char *mode)
{
  char *name;
  FILE *f;

  name=getenv("IDL_FFTW_WISDOM");

  if (name==(char*)NULL) {
    name = calloc(1024,sizeof(char));
    if (name==(char*)NULL) return (FILE*) NULL;

    strncpy(name,getenv("HOME"),1023);
    strncat(name,"/.idl_rfftw_wisdom.",1024-strlen(name));
    gethostname(name+strlen(name),1024-strlen(name));
  }
  f=fopen(name,mode);
  free(name);
  return f;
}

static char *wisdom=(char*)NULL;

void rfftw_init(void)
{
  FILE *f;
```

```
  static int done=0;

  if (done) return;

  done=1;
  f = rfftw_wisdom_file("r");
  if (f==(FILE*)NULL) {
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_INFO,"Can't read wisdom file.");
    return;
  }
  if (fftw_import_wisdom_from_file(f)!=FFTW_SUCCESS) {
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_INFO,"Bad wisdom data?");
  }
  fclose(f);

  wisdom=fftw_export_wisdom_to_string();
  IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_INFO,"Imported wisdom from file.");
}


void rfftw_finish(void)
{
  FILE *f;
  static char *new_wisdom;

  new_wisdom=fftw_export_wisdom_to_string();
  if (new_wisdom==(char*)NULL) return;

  if (wisdom!=(char*)NULL && !strcmp(wisdom,new_wisdom)) {
    fftw_free(new_wisdom);
    return;
  }

  if (wisdom!=(char*)NULL) fftw_free(wisdom);

  wisdom=new_wisdom;

  f = rfftw_wisdom_file("w");
  if (f==(FILE*)NULL) {
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_INFO,"Couldn't write wisdom file");
    return;
  }
  fftw_export_wisdom_to_file(f);
  fclose(f);
  IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_INFO,"Wrote wisdom to file");
}
```

```
/* Forward is REAL -> COMPLEX, use input dimensions for plan */
void rfftwnd_forward(IDL_VPTR in, IDL_VPTR out)
{
  fftw_real *src;
  fftw_complex *Fsrc;
  rfftwnd_plan plan;

  rfftw_init();

  src = (fftw_real*) in->value.arr->data;
  Fsrc = (fftw_complex*) out->value.arr->data;

  plan=rfftwnd_create_plan(in->value.arr->n_dim,in->value.arr- >dim,
     FFTW_REAL_TO_COMPLEX,FFTW_MEASURE|FFTW_USE_WISDOM);

  rfftwnd_one_real_to_complex(plan,src,Fsrc);
  rfftwnd_destroy_plan(plan);
  rfftw_finish();
}


/* Backward is COMPLEX -> REAL, use OUTPUT dimensions for plan! */
void rfftwnd_backward(IDL_VPTR in, IDL_VPTR out)
{
  fftw_complex *src;
  fftw_real *Fsrc;
  rfftwnd_plan plan;

  rfftw_init();

  src = (fftw_complex*) in->value.arr->data;
  Fsrc = (fftw_real*) out->value.arr->data;

  plan=rfftwnd_create_plan(out->value.arr->n_dim,out->value.arr- >dim,
     FFTW_COMPLEX_TO_REAL,FFTW_MEASURE|FFTW_USE_WISDOM);

  rfftwnd_one_complex_to_real(plan,src,Fsrc);
  rfftwnd_destroy_plan(plan);
  rfftw_finish();
}


IDL_VPTR RFFTWND(int argc, IDL_VPTR argv[])
{
  int i=0; /* Note it's use in indexing argv[i++]                */
           /* This simplifies taking away extra arguments,       */
           /* typically those specifying the number of elements  */
           /* in input arrays (available as var->value.arr->n_elts) */
```

```
IDL_VPTR a=argv[i++];    /* Input array */
IDL_VPTR odim=argv[i++], /* Output leading dim. for backward transform */

  call[1], /* For use when calling conversion routines */
  tmp;

IDL_MEMINT dim[IDL_MAX_ARRAY_DIM], tmpi;
int ndim,dir;

char odimreq[]=
  "2nd arg (leading output dimension) required for backward transform";
char odimwrong[]=
  "2nd arg (leading output dimension) has an inappropriate value";

/* TYPE CHECKING / ALLOCATION SECTION */

IDL_EXCLUDE_STRING(a);
IDL_ENSURE_ARRAY(a);
IDL_ENSURE_SIMPLE(a);

ndim = a->value.arr->n_dim; /* Shorthand */

/* If we're doing a forward transform, convert to Float,              */
/* if baclwards, convert to Complex, and make sure we have the leading */
/* dimension size, and convert it to long                             */

call[0] = a;
if (a->type!=IDL_TYP_COMPLEX && a->type!=IDL_TYP_DCOMPLEX) {

  dir = -1; /* We're doing a forward transform */
  a = IDL_CvtFlt(1,call); /* May cause a to be tmp */

} else {

  /* Require 2 arguments */
  if (argc!=2) IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_LONGJMP,odimreq);

  dir = 1;    /* Backward transform */
  a = IDL_CvtComplex(1,call); /* May cause a to be tmp */

  IDL_EXCLUDE_UNDEF(odim); /* Output leading dimension */
  IDL_ENSURE_SIMPLE(odim);
  IDL_EXCLUDE_STRING(odim);
  IDL_ENSURE_SCALAR(odim);

  call[0] = odim;
  odim = IDL_CvtLng(1,call);
```

```
  /* Check validity of the output leading dimension */
  tmpi = 2*a->value.arr->dim[0] - odim->value.l;
  if (tmpi != 1 && tmpi != 2)
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_LONGJMP,odimwrong);

  /* Give warning about overwritten input arg. */
  if (ndim>1 && !(a->flags&IDL_V_TEMP))
    IDL_Message(IDL_M_NAMED_GENERIC,IDL_MSG_INFO,"Input overwritten!");
}

/* Swap dimensions to row major */
for (i=0; i<ndim/2; i++) {
  tmpi=a->value.arr->dim[i];
  a->value.arr->dim[i] = a->value.arr->dim[ndim-i-1];
  a->value.arr->dim[ndim-i-1] = tmpi;
}

/* Copy dimensions */
for (i=0; i<a->value.arr->n_dim; i++) dim[i] = a->value.arr->dim[i];

/* Correct dimensions - for allocation */
if (dir==-1) dim[ndim-1] = 2*(dim[ndim-1]/2+1);
else         dim[ndim-1] = 2*dim[ndim-1];  /* Complex takes 2x FLOAT */

/* Storage for result */
IDL_MakeTempArray(IDL_TYP_FLOAT,a->value.arr->n_dim,dim,
    IDL_ARR_INI_INDEX,&tmp);

/* Correct output leading dimension - to make correct plan */
if (dir==1) {
  tmp->value.arr->n_elts /= tmp->value.arr->dim[ndim-1];
  tmp->value.arr->dim[ndim-1] = odim->value.l;
  tmp->value.arr->n_elts *= tmp->value.arr->dim[ndim-1];
}

if (dir==FFTW_REAL_TO_COMPLEX) rfftwnd_forward(a,tmp);
else                           rfftwnd_backward(a,tmp);

/* Swap dimensions back! */
for (i=0; i<ndim/2; i++) {
  tmpi=a->value.arr->dim[i];
  a->value.arr->dim[i] = a->value.arr->dim[ndim-i-1];
  a->value.arr->dim[ndim-i-1] = tmpi;
  tmpi=tmp->value.arr->dim[i];
  tmp->value.arr->dim[i] = tmp->value.arr->dim[ndim-i-1];
  tmp->value.arr->dim[ndim-i-1] = tmpi;
}
```

```
  i=0;

  if (a!=argv[i++]) IDL_DELTMP(a);
  if (odim!=argv[i++]) IDL_DELTMP(odim);

  if (dir==-1) {
    tmp->type = IDL_TYP_COMPLEX;  /* Change type, halve the size */
    tmp->value.arr->dim[0] /= 2;
    tmp->value.arr->n_elts /= 2;
  } else {
  }
  return tmp;
}

int IDL_Load(void)
{
  static IDL_SYSFUN_DEF func_def[] = {
    {(IDL_FUN_RET) RFFTWND,"RFFTWND",1,2}
    };
  return IDL_AddSystemRoutine(func_def,TRUE,1);
}
```