Navigation and Ancillary Information Facility

Writing a CSPICE Based Program

March 2006

Viewing This Tutorial

Navigation and Ancillary Information Facility

This coding example is an "animated" presentation that is best viewed using PowerPoint set to "Slide Show" mode.

Undefined variables are displayed in red; results are displayed in blue.

Introduction

First, let's go over the important steps in the process of writing a CSPICE-based program and putting it to work:

- **Understand the geometry problem.**
- **Identify the set of SPICE kernels that contain the data needed to perform the computation.**
- **Formulate an algorithm to compute the quantities of interest using SPICE.**
- **Write and compile the program.**
- **Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.**
- **Run the program.**

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite.   The program will compute

- **Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.**
- **Range from spacecraft to intercept point and from spacecraft to target center.**
- **Illumination angles (phase, solar incidence, and emission) at the intercept point.**

Observation Geometry

We want the boresight intercept on the surface, range from s/c to intercept and target center, and illumination angles at the intercept point.

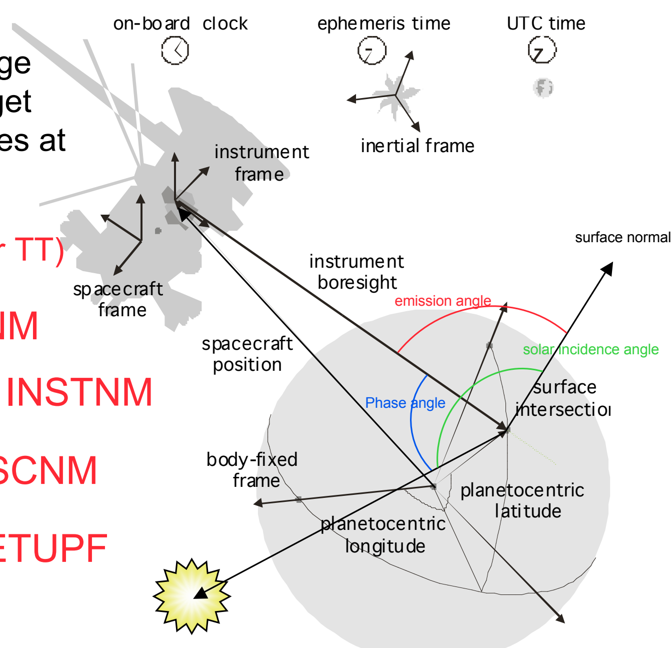
When? TIME (UTC, TDB or TT)

On what object? SATNM

For which instrument? INSTNM

For what spacecraft? SCNM

Using what model? SETUPF



Needed Data

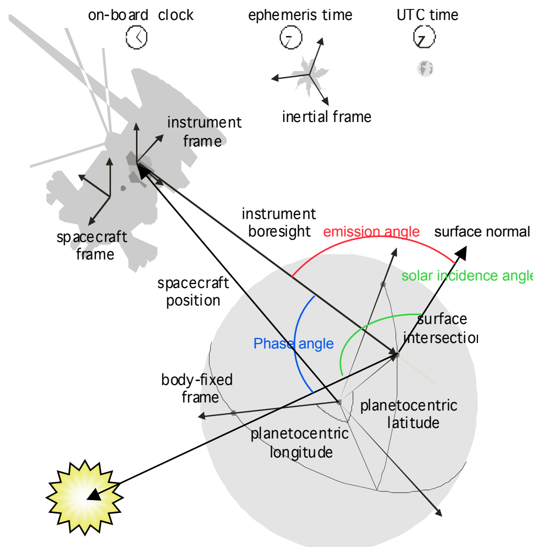
Time transformation kernels

Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,
Saturn barycenter and satellites.



Which Kinds of Kernels are Needed?

Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

Parameter	Kernel Type	File name
time conversions	generic LSK	naif0007.tls
	CASSINI SCLK	cassini.tsc
satellite orientation	generic PCK	pck00008.tpc
satellite shape	generic PCK	pck00008.tpc
satellite position	planet/sat ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	tour9201.bsp
spacecraft orientation	spacecraft CK	cas_050215.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	Instrument IK	cas_iss_v09.ti

Load Kernels

Navigation and Ancillary Information Facility

The easiest and most flexible way to make required kernels available to the program is via `furnsh_c`. For this example we make a setup file (also called a "metakernel" or "furnsh kernel") containing a list of kernels to be loaded:

```
   Note:  these kernels have been selected to support this presentation; they
   should not be assumed to be appropriate for user applications.

\begindata

   KERNELS_TO_LOAD = ( 'naif0007.tls',               'cassini.tsc',
                       'pck00008.tpc',               '020514_SE_SAT105.bsp',
                       '981005_PLTEPH-DE405S.bsp',  'tour9201.bsp',
                       'cas_050215.bc',              'cas_v37.tf',
                       'cas_iss_v09.ti'                                        )
\begintext
```

and we make the program prompt for the name of this setup file:

```
    prompt_c ( "Enter setup file name > ", FILESZ, setupf );
    furnsh_c ( setupf );
```


Programming Solution

Navigation and Ancillary Information Facility

- **Prompt for setup file ("metakernel") name; load kernels specified via setup file. (Done on previous chart.)**
- **Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via CSPICE calls.**
- **Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.**

  If there is an intersection,

  - **Convert Cartesian coordinates of the intercept point to planetocentric latitudinal and planetodetic coordinates**
  - **Compute spacecraft-to-intercept point range and spacecraft-to-target center range**
  - **Find the illumination angles (phase, solar incidence, and emission) at the intercept point**

- **Display the results.**

We discuss the geometric portion of the problem next.

Compute Surface Intercept and Ranges

Navigation and Ancillary Information Facility

Compute the intercept point (point) of the instrument boresight vector (insite) with the satellite's (satnm) surface at the TDB time of interest (et). This call also returns the distance between the spacecraft and intercept point (dist), the light-time corrected epoch at the intercept point (trgepc), the target center-to-spacecraft vector (obspos), and a boolean flag indicating whether the intercept was found (found). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface Intercept solution possible.

```
srfxpt_c ( "Ellipsoid", satnm,  et,     "CN+S",  scnm,    iframe,
           insite,      point,  &dist, &trgepc, obspos,  &found  );
```

The ranges we want are obtained from the outputs of srfxpt_c. These outputs are defined only if a surface intercept is found. If found is true, the spacecraft-to-surface intercept range is the output argument dist, and the spacecraft-to-target center range is the norm of the output argument obspos. Units are km. We use the CSPICE function vnorm_c to obtain the norm:

```
vnorm_c ( obspos )
```

We'll write out the range data along with the other program results.

Compute Lat/Lon and Illumination Angles

Navigation and Ancillary Information Facility

Compute the planetocentric latitude (pclat) and longitude (pclon), as well as the planetodetic latitude (pdlat) and longitude (pdlon) of the intersection point.

```
if ( found )
{
   reclat_c ( point,  &r,  &pclon, &pclat );

   /* Let re, rp, and f be the satellite's longer equatorial
            radius, polar radius, and flattening factor.          */
   re   =  radii[0];
   rp   =  radii[2];
   f    =  ( re - rp ) / re;

   recgeo_c ( point, re, f, &pdlon, &pdlat, &alt);
```

The illumination angles we want are the outputs of illum_c. Units are radians. For this call, normal light time and stellar aberration corrections suffice.

```
   illum_c ( satnm,  et,      "LT+S",  scnm,
             point,  &phase,  &solar,  &emissn );
```

Geometry Calculations: Summary

Navigation and Ancillary Information Facility

```
/* Compute the boresight ray intersection with the surface of the
   satellite. `dist' and vnorm_c(obspos) yield desired ranges.       */

srfxpt_c ( "Ellipsoid", satnm, et,     "CN+S",  scnm,    iframe,
           insite,      point,  &dist, &trgepc, obspos,  &found );

/* If an intercept is found, compute planetocentric and planetodetic
   latitude and longitude of the point.                               */
if ( found )
{
   reclat_c ( point, &r, &pclon, &pclat );
   /* Let re, rp, and f be the satellite's longer equatorial
      radius, polar radius, and flattening factor.                  */
   re   =   radii[0];
   rp   =   radii[2];
   f    =   ( re - rp ) /  re;
   recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );

   /* Compute illumination angles at the surface point.             */
   illum_c ( satnm, et, "LT+S", scnm, point, &phase, &solar, &emissn );
   ...
}
else
{
   ...
}
```


Get Inputs - 1

Navigation and Ancillary Information Facility

The code above used quite a few inputs that we don't have yet:

- **TDB epoch of interest (`et`);**
- **satellite and s/c names (`satnm, scnm`);**
- **satellite ellipsoid radii (`radii`);**
- **instrument fixed frame name (`iframe`);**
- **instrument boresight vector in the instrument frame (`insite`);**

Some of these values are user inputs; others can be obtained via CSPICE calls once the required kernels have been loaded.

Let's prompt for the satellite name (`satnm`), spacecraft name (`scnm`), instrument name (`instnm`) and time of interest (`time`):

```
prompt_c ( "Enter satellite name  > ", WORDSZ, satnm  );
prompt_c ( "Enter spacecraft name > ", WORDSZ, scnm   );
prompt_c ( "Enter instrument name > ", WORDSZ, instnm );
prompt_c ( "Enter time            > ", WORDSZ, time   );
```

Get Inputs - 2

Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from CSPICE calls:

To get the TDB epoch (et) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

```
str2et_c ( time, &et );
```

To get the satellite's ellipsoid radii (radii):

```
bodvrd_c ( satnm, "RADII", 3, &i, radii );
```

To get the instrument boresight direction (insite) and the name of the instrument frame (iframe) in which it is defined:

```
bodn2c_c ( instnm, &instid, &found );

 if ( !found )
 {
    setmsg_c ( "Instrument name # could not be "
               "translated to an ID code."      );
    errch_c  ( "#",  instnm                      );
    sigerr_c ( "NAMENOTFOUND"                    );
 }
 getfov_c ( instid, ROOM,    WORDSZ,  WORDSZ,
            shape,  iframe,  insite,  &n,     bundry );
```


Getting Inputs: Summary

Navigation and Ancillary Information Facility

```
/* Prompt for the user-supplied inputs for our program      */
   prompt_c ( "Enter setup file name > ", FILESZ, setupf );
   furnsh_c ( setupf );
   prompt_c ( "Enter satellite name  > ", WORDSZ, satnm  );
   prompt_c ( "Enter spacecraft name > ", WORDSZ, scnm   );
   prompt_c ( "Enter instrument name > ", WORDSZ, instnm );
   prompt_c ( "Enter time            > ", WORDSZ, time   );

/* Get the epoch corresponding to the input time:  */
   str2et_c ( time, &et );

/* Get the radii of the satellite. */
   bodvrd_c ( satnm, "RADII", 3, &i, radii );

/* Get the instrument boresight and frame name. */

   bodn2c_c ( instnm, &instid, &found );
   if ( !found )
   {
        setmsg_c ( "Instrument name # could not be "
                   "translated to an ID code."      );
        errch_c  ( "#",  instnm                      );
        sigerr_c ( "NAMENOTFOUND"                    );
   }
   getfov_c ( instid, ROOM,    WORDSZ,  WORDSZ,
              shape,  iframe,  insite,  &n,     bundry );
```

Display Results

Navigation and Ancillary Information Facility

```
    /* Display results.  Convert angles from radians to degrees for output. */
    printf ( "\n"
             "Intercept planetocentric longitude       (deg):   %11.6f\n"
             "Intercept planetocentric latitude        (deg):   %11.6f\n"
             "Intercept planetodetic longitude         (deg):   %11.6f\n"
             "Intercept planetodetic latitude          (deg):   %11.6f\n"
             "Range from spacecraft to intercept point  (km):   %11.6f\n"
             "Range from spacecraft to target center    (km):   %11.6f\n"
             "Intercept phase angle                    (deg):   %11.6f\n"
             "Intercept solar incidence angle          (deg):   %11.6f\n"
             "Intercept emission angle                 (deg):   %11.6f\n",
             dpr_c() * pclon,
             dpr_c() * pclat,
             dpr_c() * pdlon,
             dpr_c() * pdlat,
             dist,
             vnorm_c( obspos ),
             dpr_c() * phase,
             dpr_c() * solar,
             dpr_c() * emissn
    );
}
else
{
   printf ( "No intercept point found at %s\n", time );
}
```


Complete the Program

Navigation and Ancillary Information Facility

To finish up the program we need to declare the variables we've used.

- **We'll highlight techniques used by NAIF programmers**
- **Add remaining C code required to make a syntactically valid program**

Complete Source Code - 1

Navigation and Ancillary Information Facility

```
#include <stdio.h>
#include "SpiceUsr.h"

int main ()
{
    #define  FILESZ          256
    #define  WORDSZ          41
    #define  ROOM            10


    SpiceBoolean    found;

    SpiceChar       iframe[WORDSZ];
    SpiceChar       instnm[WORDSZ];
    SpiceChar       satnm [WORDSZ];
    SpiceChar       scnm  [WORDSZ];
    SpiceChar       setupf[FILESZ];
    SpiceChar       shape [WORDSZ];
    SpiceChar       time  [WORDSZ];

    SpiceDouble     alt;
    SpiceDouble     bundry[ROOM][3];
    SpiceDouble     dist;
    SpiceDouble     emissn;
    SpiceDouble     et;
    SpiceDouble     f;
    SpiceDouble     insite[3];
    SpiceDouble     obspos[3];
    SpiceDouble     pclat;
    SpiceDouble     pclon;
    SpiceDouble     pdlat;
    SpiceDouble     pdlon;
    SpiceDouble     phase;
    SpiceDouble     point [3];
    SpiceDouble     r;
    SpiceDouble     radii [3];
    SpiceDouble     re;
    SpiceDouble     rp;
    SpiceDouble     solar;
    SpiceDouble     trgepc;

    SpiceInt        i;
    SpiceInt        instid;
    SpiceInt        n;
```


Complete Source Code - 2

Navigation and Ancillary Information Facility

```
/* Prompt for the user-supplied inputs for our program      */
   prompt_c ( "Enter setup file name > ", FILESZ, setupf );
   furnsh_c ( setupf );
   prompt_c ( "Enter satellite name  > ", WORDSZ, satnm  );
   prompt_c ( "Enter spacecraft name > ", WORDSZ, scnm   );
   prompt_c ( "Enter instrument name > ", WORDSZ, instnm );
   prompt_c ( "Enter time            > ", WORDSZ, time   );

/* Get the epoch corresponding to the input time:  */
   str2et_c ( time, &et );

/* Get the radii of the satellite. */
   bodvrd_c ( satnm, "RADII", 3, &i, radii );

/* Get the instrument boresight and frame name. */

   bodn2c_c ( instnm, &instid, &found );
   if ( !found )
   {
        setmsg_c ( "Instrument name # could not be "
                   "translated to an ID code."      );
        errch_c  ( "#",  instnm                     );
        sigerr_c ( "NAMENOTFOUND"                   );
   }
   getfov_c ( instid, ROOM,    WORDSZ,   WORDSZ,
              shape,   iframe,  insite,  &n,     bundry );
```

Complete Source Code - 3

Navigation and Ancillary Information Facility

```
/* Compute the boresight ray intersection with the surface of the
   satellite. `dist' and vnorm_c(obspos) yield desired ranges.      */
srfxpt_c ( "Ellipsoid", satnm,  et,    "CN+S",  scnm,    iframe,
           insite,      point,  &dist, &trgepc, obspos,  &found );
/* If an intercept is found, compute planetocentric and planetodetic
   latitude and longitude of the point.                              */
if ( found )
{
   reclat_c ( point, &r, &pclon, &pclat );
   /* Let re, rp, and f be the satellite's longer equatorial
   radius, polar radius, and flattening factor.                      */
   re    =   radii[0];
   rp    =   radii[2];
   f     =   ( re - rp ) /  re;
   recgeo_c ( point, re, f, &pdlon, &pdlat, &alt );

   /* Compute illumination angles at the surface point.
*/    illum_c ( satnm, et, "LT+S", scnm, point, &phase, &solar, &emissn
);
   /* Display results.  Convert angles from radians to degrees
      for output.                                                    */
   printf ( "\n"
                      "Intercept planetocentric longitude       (deg):
                                 %11.6f\n"
                      "Intercept planetocentric latitude        (deg):
                                 %11.6f\n"
```


Complete Source Code - 4

Navigation and Ancillary Information Facility

```
        "Intercept planetodetic longitude         (deg):   %11.6f\n"
        "Intercept planetodetic latitude          (deg):   %11.6f\n"
        "Range from spacecraft to intercept point  (km):   %11.6f\n"
        "Range from spacecraft to target center    (km):   %11.6f\n"
        "Intercept phase angle                    (deg):   %11.6f\n"
        "Intercept solar incidence angle          (deg):   %11.6f\n"
        "Intercept emission angle                 (deg):   %11.6f\n",
        dpr_c() * pclon,
        dpr_c() * pclat,
        dpr_c() * pdlon,
        dpr_c() * pdlat,
        dist,
        vnorm_c( obspos ),
        dpr_c() * phase,
        dpr_c() * solar,
        dpr_c() * emissn
   );}
   else {
      printf ( "No intercept point found at %s\n", time );
   }
   return(0);
}
```

Compile and Link the Program - 1

Navigation and Ancillary Information Facility

- First be sure that both the CSPICE Toolkit and a C compiler are properly installed.
  - A "hello world" C program must be able to compile, link, and run successfully in your environment.
  - Any of the mkprodct.* scripts in the cspice/src/* paths of the CSPICE installation should execute properly.
- Ways to compile and link the program:
  - If you're familiar with the "make" utility, create a makefile. Use compiler and linker options from the mkprodct.* script found in the cspice/src/cook_c path of your CSPICE installation.
  - Or, modify the cookbook mkprodct.* build script.
    - » Your program name must be *.pgm, for example demo.pgm, to be recognized by the script.
    - » Change the library references in the script to use absolute pathnames.
    - » Change the path for the executable to the current working directory.
    - » If you compiler supports it, add a –I option to reference the cspice/include path to make CSPICE *.h files available. Otherwise, copy those files from the include path to your current working directory.
    - » On some platforms, you must modify the script to refer to your program by name.

Compile and Link the Program - 2

Navigation and Ancillary Information Facility

- Or, compile the program on the command line. The program must be linked against the CSPICE object library cspice.a (cspice.lib under MS Visual C++/C) and the C math library. On a PC running Linux and gcc, if
  - » The gcc compiler is in your path
    - As indicated by the response to the command "which gcc"
  - » the Toolkit is installed in the path (for the purpose of this example) /myhome/cspice
  - » You've named the program demo.c

  then you can compile and link your program using the command

  »
```
gcc -I/myhome/cspice/include \
    -o demo \
     demo.c  /myhome/cspice/lib/cspice.a -lm
```
  - Note: the preprocessor flag

    `-DNON_UNIX_STDIO`

    used in the mkprodct.csh script is needed for code generated by f2c, but is usually unnecessary for compiling user code.

Compile and Link the Program - 3

Terminal Window

```
Prompt> mkprodct.csh

       Setting default compiler:
gcc

       Setting default compile options:
       -c -ansi -O2 -DNON_UNIX_STDIO

       Setting default link options:
       -lm

       Compiling and linking:  demo.pgm
Compiling and linking:  demo.pgm

Prompt>
```


Running the Program - 1

It looks like we have everything taken care of:

- **We have all necessary kernels**
- **We made a setup file (metakernel) pointing to them**
- **We wrote the program**
- **We compiled and linked it**

Let's run it.

Running the Program - 2

Terminal Window

```
Prompt> demo
Enter setup file name    > setup.ker
Enter satellite name     > titan
Enter spacecraft name    > cassini
Enter instrument name    > cassini_iss_nac
Enter time               > 2005 feb 15 8:15 UTC

Intercept planetocentric longitude       (deg):   -156.443003
Intercept planetocentric latitude        (deg):     18.788539
Intercept planetodetic longitude         (deg):   -156.443003
Intercept planetodetic latitude          (deg):     18.788539
Range from spacecraft to intercept point  (km):   4810.941881
Range from spacecraft to target center    (km):   7384.326555
Intercept phase angle                    (deg):     43.274588
Intercept solar incidence angle          (deg):     41.038424
Intercept emission angle                 (deg):      2.514613
Prompt>
```


Backup

Navigation and Ancillary Information Facility

- **Latitude definitions:**
  - **Planetocentric latitude of a point P: angle between segment from origin to point and x-y plane (red arc in diagram).**
  - **Planetodetic latitude of a point P: angle between x-y plane and extension of ellipsoid normal vector N that connects x-y plane and P (blue arc in diagram).**