Department of Geomatic Engineering, University College London

Analytical Solar Radiation Pressure and Thermal Re-radiation Modelling Software

Version 5.05

User Manual

M. Y. Gulamali and M. K. Ziebart

29 September 2006

Department of Geomatic Engineering
University College London
Gower Street
London
WC1E 6BT
United Kingdom

Contents

1 Introduction

1.1 Description

Software to simulate the non-conservative forces acting upon a spacecraft has been developed by members of the Department of Geomatic Engineering [3] at University College London. This software may be used to compute the accelerations upon a complicated spacecraft structure due to solar radiation pressure (SRP) and thermal re-radiation (TRR), as a function of the orientation of the Sun in the spacecraft's frame of reference. It may also be used to determine the area profile of a spacecraft from different directions.

The analytical SRP and TRR modelling software described in this manual simulates a source of irradiance (i.e. the Sun) as a pixel array. The pixel array is oriented at a range of locations around the spacecraft depending on the specified orientation scheme (see Section 3.2). A ray tracing algorithm is then used to determine if light rays from individual pixels in the array strike the spacecraft. If spacecraft components are found to be irradiated by the light rays then their contribution to the overall SRP and/or TRR acceleration is computed. Secondary reflections and shadowing by features of the spacecraft is also taken into account. Thus the resultant accelerations upon a spacecraft due to SRP and/or TRR forces are determined for the entire range of orientations of the pixel array, and are output to a comma delimited ASCII file. Readers are referred to Ziebart (2004) [19] and Adhya (2005) [1] for a more comprehensive account of the algorithms used to model the SRP and TRR accelerations in this software. The fidelity of earlier versions of the software has been discussed by Ziebart *et al.* (2003) [18].

The algorithm used to determine the area profile of a spacecraft is similar to that above, however, instead of computing the accelerations due to SRP and/or TRR, the software simply counts the number of pixels illuminating the spacecraft for a given orientation of the pixel array. This is then scaled by the size of each pixel to determine the area profile of the spacecraft.

This document provides a manual for the modelling software, describing how to configure, compile and execute it on a range of computational platforms.

1.2 Block modelling

This version of the analytical SRP and TRR modelling software has been optimised using the block modelling approach first introduced in version 4.00. Block modelling reduces the execution time of the software by grouping components together into "blocks". Subsequently, during the ray tracing phase of the algorithm, if a block is not intersected by a ray of light from the pixel array, then the entire group of components associated with that block may be considered to have not been intersected by the ray of light, and consequently will not be irradiated. For large spacecraft structures, it has been found that block modelling can reduce the execution time by at least a half. Block modelling is described in more detail by Sibthorpe (2006) [16], and may be switched on or off at the compilation stage (see Section 2.3).

1.3 Parallelisation

The analytical SRP and TRR modelling software is capable of being run on multiprocessor computing resources supporting the Message Passing Interface (MPI) Version 1 [7]. Parallelisation is achieved by concurrently computing the SRP and TRR forces upon a spacecraft for a range of orientations of the pixel array. In general, this serves to reduce the overall execution time. However, due to the nature of the parallelisation scheme, this may not be practical when the spacecraft structure consists of many components (> 1000) and the computing resource has few processors (< 32), because communication between processors will then overcome any advantage of concurrent execution. Parallelisation may be switched on or off during the compilation stage (see Section 2.3).

2 Compilation

2.1 Introduction

The analytical SRP and TRR modelling software is written in C++, consequently a C++ compiler is required in order to compile it from the source code. The MPI Version 1 libraries are also required to compile and execute the parallelised version of the software. The following subsections describe how to configure and compile the software for a number of different operating environments (i.e. Microsoft Windows and UNIX based machines).

2.2 Organisation of the software

The source code for the analytical SRP and TRR modelling software is organised into a number of subdirectories under the parent directory as follows:

Directory	**Contents**
`include`	C++ definitions and header files.
`parameters`	Example parameter definition files.
`src`	C++ source files.
`userfiles`	Example spacecraft description files.

A Makefile also resides in the parent directory. This is required to compile the software under a UNIX based operating system using the GNU Make [6] utility (see Section 2.5).

2.3 Compilation options

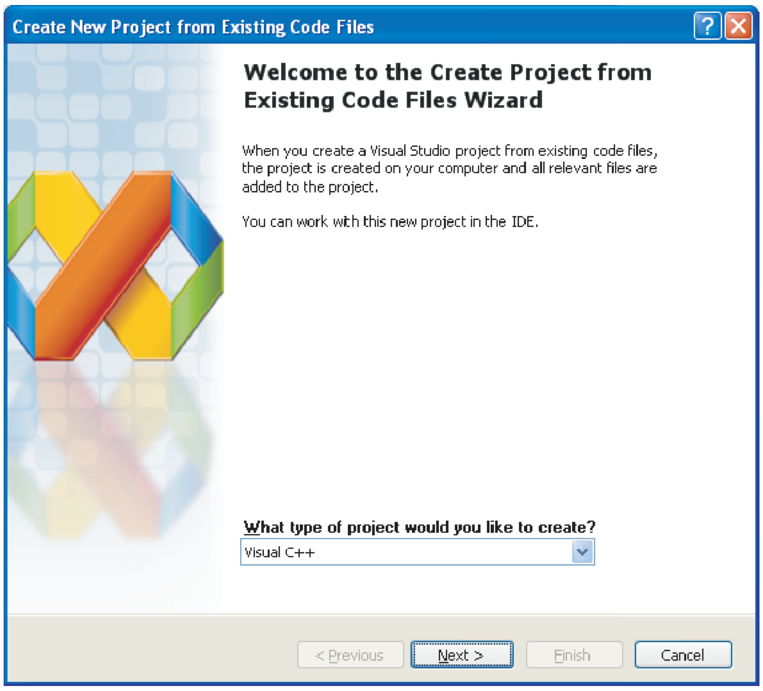
Prior to compilation, a number of parameters may be adjusted to control various aspects of the algorithms used in the modelling software. These parameters may be found in the `include/definitions.h` header file. They are described in more detail in Table 1 in Appendix A.

2.4 Microsoft Visual Studio

The following instructions explain how the analytical SRP and TRR modelling software may be compiled in Microsoft Visual Studio 2005 [10] on the Microsoft Windows XP platform.

1. Begin Microsoft Visual Studio 2005 and choose the menu option to start a new project from existing code (*File* → *New* → *Project from Existing Code...*)

2. In the wizard choose to create a Visual C++ project and press the *Next* button i.e.,



Create New Project from Existing Code Files

Welcome to the Create Project from Existing Code Files Wizard

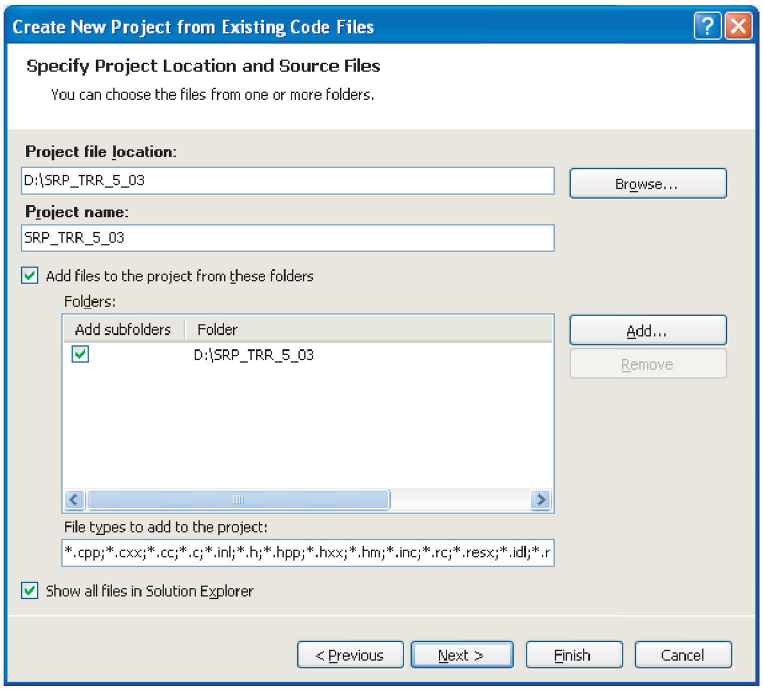
When you create a Visual Studio project from existing code files, the project is created on your computer and all relevant files are added to the project.

You can work with this new project in the IDE.

What type of project would you like to create?

Visual C++

3. Enter the location of the parent directory of the analytical SRP and TRR modelling software, and give the new project a unique name. Ensure the *Add subfolders* option is chosen and then click the *Next* button i.e.,



Create New Project from Existing Code Files

Specify Project Location and Source Files

You can choose the files from one or more folders.

Project file location: D:\SRP_TRR_5_03

Project name: SRP_TRR_5_03

☑ Add files to the project from these folders

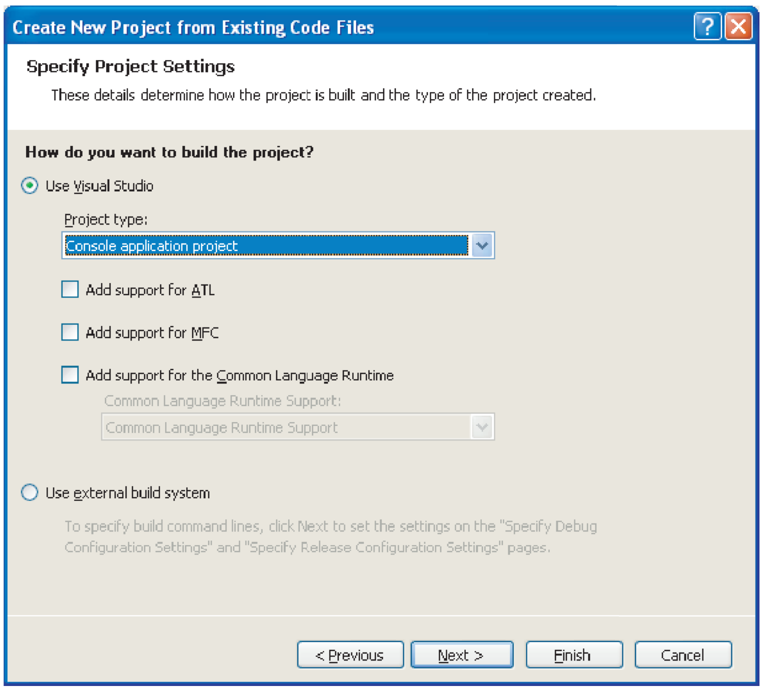
Folders:

Add subfolders	Folder
✓	D:\SRP_TRR_5_03

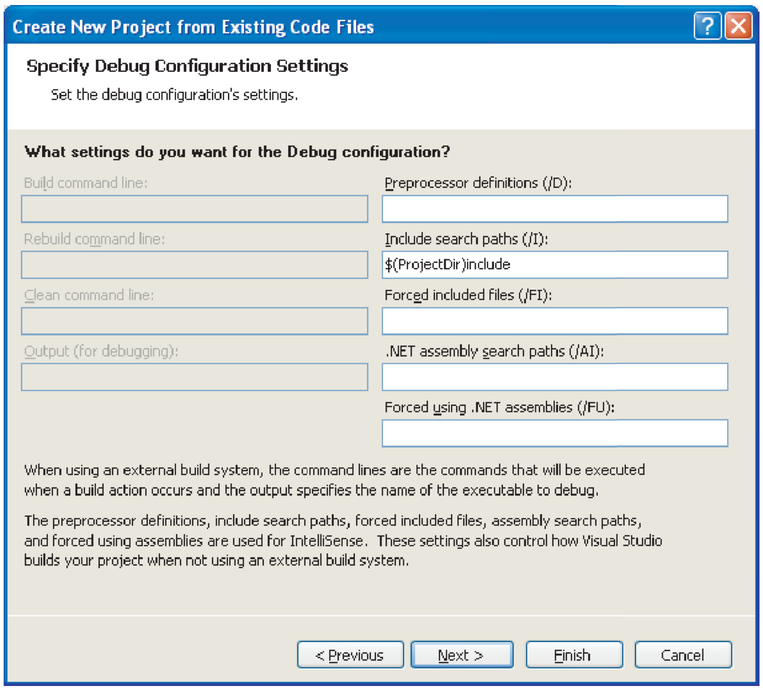
File types to add to the project: *.cpp;*.cxx;*.cc;*.c;*.inl;*.h;*.hpp;*.hxx;*.hm;*.inc;*.rc;*.resx;*.idl;*.r

☑ Show all files in Solution Explorer

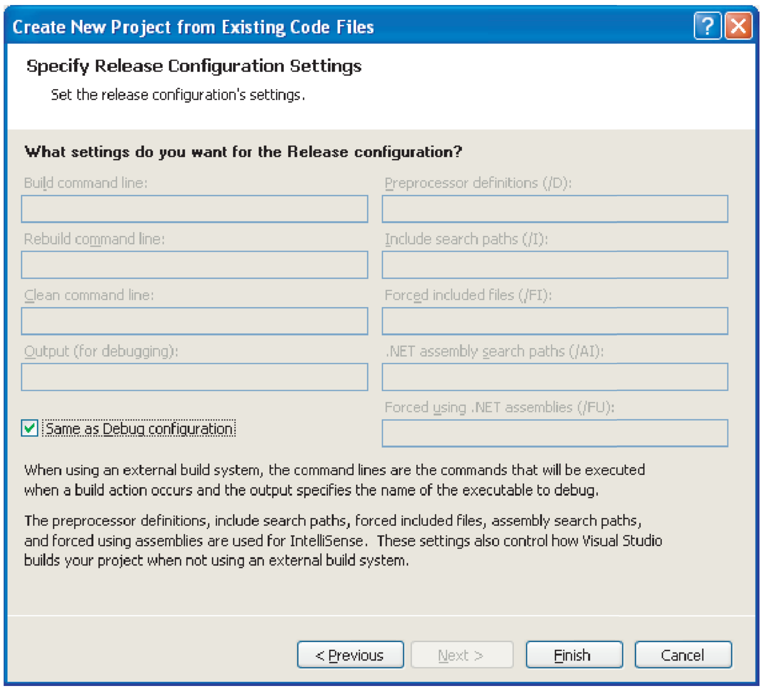
4. Choose the *Console application project* option for the project type and press the *Next* button i.e.,


Create New Project from Existing Code Files
Specify Project Settings
These details determine how the project is built and the type of the project created.
How do you want to build the project?
Use Visual Studio
Project type:
Console application project
Add support for ATL
Add support for MFC
Add support for the Common Language Runtime
Common Language Runtime Support:
Common Language Runtime Support
Use external build system
To specify build command lines, click Next to set the settings on the "Specify Debug Configuration Settings" and "Specify Release Configuration Settings" pages.
< Previous
Next >
Finish
Cancel

5. In the *Include Search Paths* add `$(ProjectDir)include`. Then press the *Next* button,


Create New Project from Existing Code Files
Specify Debug Configuration Settings
Set the debug configuration's settings.
What settings do you want for the Debug configuration?
Build command line:
Preprocessor definitions (/D):
Rebuild command line:
Include search paths (/I):
$(ProjectDir)include
Clean command line:
Forced included files (/FI):
Output (for debugging):
.NET assembly search paths (/AI):
Forced using .NET assemblies (/FU):
When using an external build system, the command lines are the commands that will be executed when a build action occurs and the output specifies the name of the executable to debug.
The preprocessor definitions, include search paths, forced included files, assembly search paths, and forced using assemblies are used for IntelliSense. These settings also control how Visual Studio builds your project when not using an external build system.
< Previous
Next >
Finish
Cancel

6. Ensure that the *Same as Debug configuration* option is ticked and then press the *Finish* button,


Create New Project from Existing Code Files

Specify Release Configuration Settings

Set the release configuration's settings.

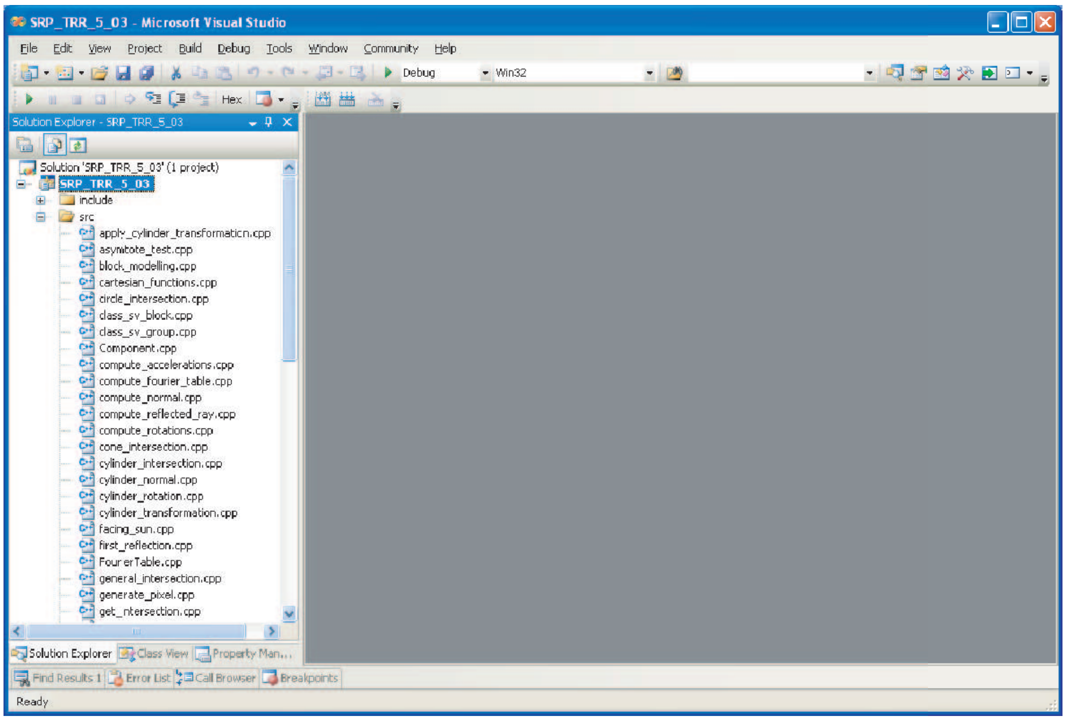
What settings do you want for the Release configuration?

Build command line:	Preprocessor definitions (/D):
Rebuild command line:	Include search paths (/I):
Clean command line:	Forced included files (/FI):
Output (for debugging):	.NET assembly search paths (/AI):
[✓] Same as Debug configuration	Forced using .NET assemblies (/FU):

When using an external build system, the command lines are the commands that will be executed when a build action occurs and the output specifies the name of the executable to debug.

The preprocessor definitions, include search paths, forced included files, assembly search paths, and forced using assemblies are used for IntelliSense. These settings also control how Visual Studio builds your project when not using an external build system.

7. Visual Studio will now create a new project with the appropriate settings to build a serial version of the analytical SRP and TRR modelling software, e.g.,



8. Open the `definitions.h` file in the include folder to configure the software (see Section 2.3). In particular be sure to set the `USE_MPI` definition to build a

serial or parallelised version of the software. Refer to Appendix C for details about incorporating MPI into the software to create a parallelised version.

9. Compile the software using the build option in the menu (*Build* → *Build Solution*) or by pressing the F7 key.

Upon compilation, the analytical SRP and TRR modelling software should reside as a single executable file, `SRP_TRR_5_05.exe`, in the `Debug` or `Release` directory (depending on the build configuration chosen). Information about running this executable is given in Section 3.

2.5 UNIX based compilers

A `Makefile` has been provided in the source directory of the analytical SRP and TRR modelling software in order to aid compilation on UNIX based computing resources using the GNU Make utility [6]. To compile the software:

1. Open the `definitions.h` file in the `include` folder to configure the software (see Section 2.3).
2. If a parallelised version of the software is required, please ensure that the appropriate version (specific to the architecture of your system) of MPI is installed. Refer to your local UNIX Guru if you are unsure.
3. Open the `Makefile` in the source directory in your favourite text editor.
4. Edit the `BINDIR` variable to refer to the directory where you wish the binary executable to be created.
5. Edit the `MPIHOME` variable to refer to the path of the MPI distribution on your system.
6. Edit the `CC` variable to refer to the MPI enabled C++ compiler on your system.
7. Edit the `INCLUDE`, `LIBS` and `FLAGS` variables to use any appropriate flags for your specific compiler and MPI distribution.
8. Saved the edited `Makefile`, exit your editor and type “make” in your terminal window in the source directory in order to compile the software.

Upon compilation, the analytical SRP and TRR modelling software should reside as a single executable file, `srp_trr_5_05`, in the `BINDIR` directory chosen in the `Makefile`. A “make clean” option has also been provided to remove intermediate object files, and a “make distclean” option has been provided to also remove executable binary files, allowing for redistribution of the software. Information about running the analytical SRP and TRR modelling software executable is given in the following section.

3 Usage

3.1 Introduction

The analytical SRP and TRR modelling software requires several command line arguments in order to run successfully, e.g. on MS-DOS based platforms,

```
SRP_TRR_5_05.exe <parameter file> <spacecraft file> <output file>
```

where the arguments refer to the full path of a parameter file, a spacecraft description file and an output file, respectively. Parallelised versions of the software will require the `mpirun` command [11] to run on multiple processors, e.g. on UNIX based multiprocessor platforms,

```
mpirun -np <procs> srp_trr_5_05 <parameter file> <spacecraft file> <output file>
```

where the `procs` argument refers to the number of processors to be used. The remainder of this section describes the content and format of the parameter file, the spacecraft description file and the output file in more detail. Appendix D describes how to execute the software on high performance computing resources maintained at UCL.

3.2 Pixel array orientation schemes

One of the parameters that must be set for the execution of the analytical SRP and TRR modelling software is the pixel array orientation scheme. The scheme describes how the pixel array is orientated around the spacecraft during successive iterations of the force modelling algorithms. This version of the software incorporates two different orientation schemes which are chosen and configured through the parameter file (see Section 3.3).

3.2.1 Earth-Probe-Sun angle range

This pixel array orientation scheme computes the SRP and TRR accelerations upon a spacecraft as a function of the Earth-Probe-Sun (EPS) angle. This is defined as the angle between the Earth and the Sun in the spacecraft body-fixed-frame, as illustrated in Figure 1. Thus this scheme orients the source of luminance (i.e. the Sun) in the Z-X plane, rotating it around the positive Y-axis with successive

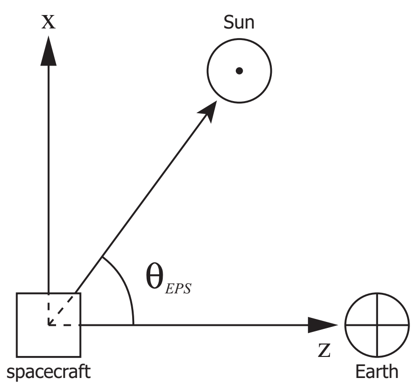


Figure 1: Earth-Probe-Sun (EPS) angle, θ_{EPS}, defined as the angle between the Earth and the Sun in the spacecraft body-fixed-frame. Here, the X-axis points along track, the Y-axis points across track and the Z-axis points towards the Earth.

iterations of the modelling algorithms. The starting angle, increment angle, and final angle associated with the rotations, must be specified in the parameter file.

It is possible to simulate a source of luminance rotating around the spacecraft in a plane other than the Z-X plane by applying a perturbation to this orientation scheme. The perturbation acts to reorient a quaternion axis defined to be perpendicular to the Sun-spacecraft vector, and in the Z-X plane, by a specified perturbation angle, δ, as shown in Figure 2. Readers are referred to Ziebart (2004) [19] for more information about the calculations involved. The perturbation may be performed by selecting the appropriate switch in the parameter file and specifying the perturbation angle (see Section 3.3).

3.2.2 Sphere points algorithm

The sphere points orientation scheme positions the pixel array around the spacecraft using an algorithm described by Saff and Kuijlaars (1997) [14]. This involves optimising the range of orientations of the pixel array to cover the entire 4π steradians around the spacecraft (see Figure 3). Consequently, this scheme allows one to compute the SRP and TRR accelerations around an entire spacecraft, in 3-dimensions, in a highly efficient manner. The total number of pixel array orientations required, as well as the start and end indices (see [14]) must be specified in the parameter file in order to use this scheme. In general, the total number of orientations, N, is related to the geodesic distance, s, between points on a sphere of unit radius as:

$$N = \left(\frac{3.6}{s}\right)^2$$

where the value of the nominator was chosen through numerical experimentation by Saff and Kuijlaars (1997) [14]. This value may be altered by setting the `SP_CONST` definition in the definitions file (see Section 2.3).

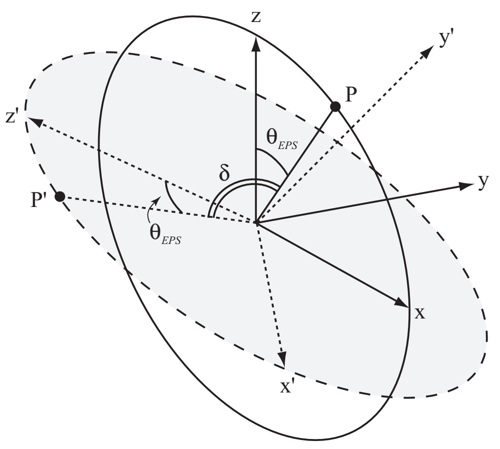


Figure 2: Orientation of pixel array under the perturbed EPS angle range orientation scheme. The pixel array, P, is rotated through the quaternion axis by an angle of δ, to P'. Axes marked with primes denote the perturbed frame of reference. The shaded plane denotes the perturbed plane of EPS angle rotations.

3.3 The parameter file

The parameter file required by the analytical SRP and TRR modelling software consists of an ASCII file defining the parameters of a run. Each line of the parameter file consists of a keyword and a value as follows:

```
keyword = value
```

where at least a single character space must be present on either side of the equality sign. Comments may be added to the file by beginning the comment line with two consecutive forward slashes (i.e. "//"). These may also be used to add comments after a keyword-value pair i.e.,

```
keyword = value // comment
```

however, their interpretation by the software is specific to the compiler that is used and consequently this syntax for commenting is not recommended. Table 2 in Appendix A outlines the keywords used by the current version of the modelling software, and the format of their associated values. They may be written in any order in the parameter file, and some may even be omitted depending on the type of pixel array orientation scheme chosen. Example parameter files may be found in the `parameters` subdirectory of the software distribution.

3.4 The spacecraft description file

The spacecraft description file required by the analytical SRP and TRR modelling software consists of an ASCII file describing the geometrical dimensions and physical characteristics of components of the spacecraft. Several types of components may

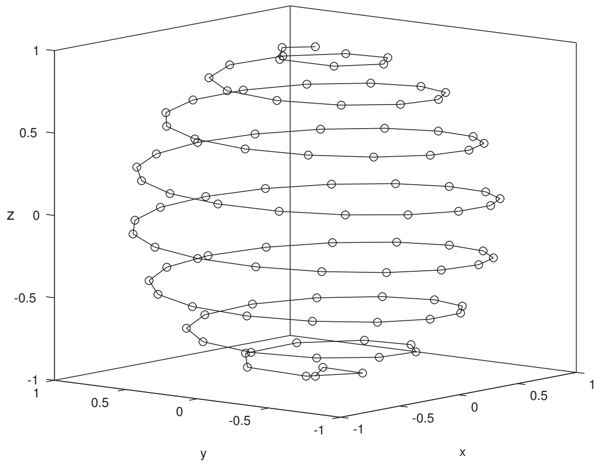


Figure 3: The method of Saff and Kuijlaars (1997) [14] serves to distribute points equidistant apart on the surface of a sphere. Here $N = 100$, circles represent the points and the solid line traces out the spiral pattern achieved.

be modelled and the subsections below describe how they are represented in the spacecraft description file. The coordinate system used in the file corresponds to the Body-Fixed system (BFS) (e.g. see Figure 1).

Each component record in the spacecraft description file begins with a header line specifying the type of component being described by that record (e.g. `GENERAL`, `SPHERE`, `CONEI` etc.). Thereafter, there is a comment line consisting of a record number, two solidi (//), a material type number, a component group number, and possibly a comment about the component, e.g.,

```
3 // 001 0010 main bus +Z face
```

In this example the record number is 3; the material type number is 1 (any leading zeros are ignored); the component group number is 10; and, the comment provides some information about the component.

The material type number is required for the TRR algorithms and indicates whether or not the surface of the component is covered in multi-layer insulation (MLI), where:

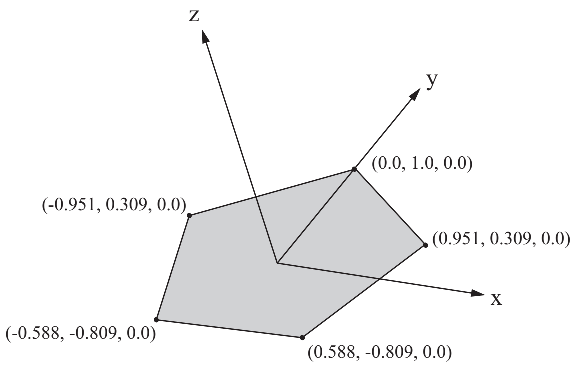
0	=	component **is not** covered in MLI
1	=	component **is** covered in MLI

The component group number indicates the group of components to which this component belongs. This is used to reduce the runtime of the analytical SRP and TRR modelling software using the block modelling approach (see Section 1.2).

Consecutive lines of the component record specify geometrical attibutes of the component and thus are dependent on the geometrical type of the component. They are described in more detail in the following subsections.

```
GENERAL
4 // 001 0001 pentagon covered in MLI
5
 0.0    1.0   0.0
-0.951  0.309 0.0
-0.588 -0.809 0.0
 0.588 -0.809 0.0
 0.951  0.309 0.0
0.1 0.5
```

(a)



(b)

Figure 4: A regular pentagon covered in MLI. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a schematic of the component. The surface normal of this component is in the positive z-direction.

The final line of each component record specifies the reflectivity and specularity coefficients, respectively, of the surface material of the component.

Example spacecraft description files may be found in the `userfiles` subdirectory of the software distribution.

3.4.1 Planar polygons

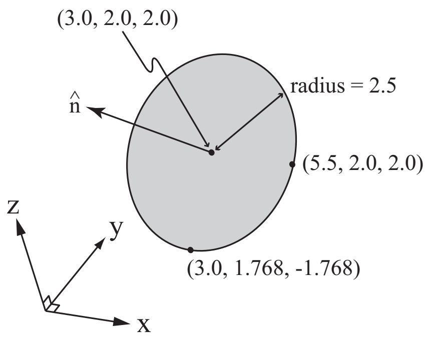
Planar polygon components consist of planar, regular or irregular, polygons with upto `CARTESIANS_IN_POLY` sides (see Table 1). They are described as a component record with the header: `GENERAL`. The number of vertices of the polygon is specified on a line after the comment line. Thereafter, a list of the positions of each of those vertices is given, where the list proceeds in the anti-clockwise direction as one looks down the normal onto the surface. This condition ensures that the calculation of the surface normal is consistent with the same calculation for the other geometrical shapes. Figure 4 shows an example of the representation of a regular pentagon component in the spacecraft description file. Here, the surface normal is in the positive z-direction.

3.4.2 Planar circles

Planar circle components are described in the spacecraft description file with the `CIRCLE` header. Following the comment line, the radius of the circle is specified. Thereafter the positions of the centre of the circle and two points on the circumference are given. The points on the circumference are listed in an order similar to that for the planar polygon components. An example is shown in Figure 5.

```
CIRCLE
5 // 000 0001 shiny metal circle
2.5
3.0 2.0   2.0
3.0 1.768 1.768
5.5 2.0   2.0
0.8 0.8
```

(a)

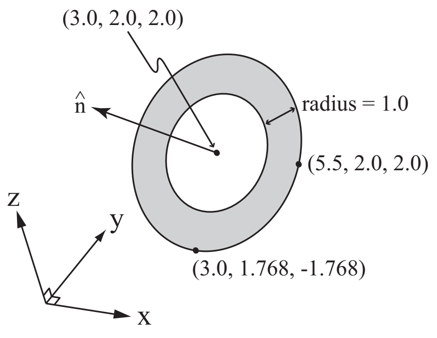


(b)

Figure 5: A highly reflective and highly specular circle component of radius 2.5 units, centred at $(3.0, 2.0, 2.0)$ in the frame of reference. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a schematic of the component. The vector $\hat{n}$ points in the direction of the surface normal.

```
RING
6 // 000 0001 shiny metal ring
2.5
1.5
3.0 2.0   2.0
3.0 1.768 1.768
5.5 2.0   2.0
0.8 0.8
```

(a)



(b)

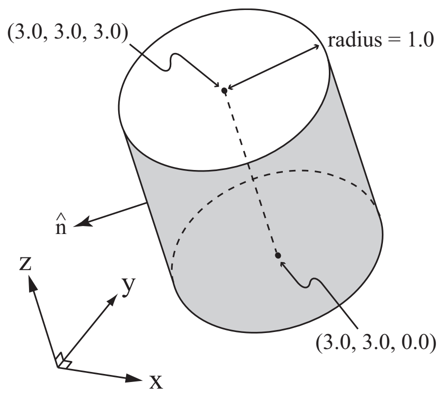
Figure 6: A highly reflective and highly specular ring component centered at $(3.0, 2.0, 2.0)$ in the frame of reference. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a schematic of the component. The vector $\hat{n}$ points in the direction of the surface normal.

3.4.3 Planar rings

Planar ring components (or annulus components) are described in the spacecraft description file with the `RING` header. The remainder of the component is specified in a similar manner to the planar circle component but an extra line is added after the radius of the outter circle forming the ring is specified. This line specifies the inner radius of the ring (see Figure 6).

```
CYL_X
7 // 000 0001 dull plastic cylinder
1.0
3.0 3.0 0.0
3.0 3.0 3.0
0.4 0.1
```

(a)



(b)

Figure 7: A cylinder component. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a diagram of the component. The vector $\hat{n}$ points in the direction of the surface normal.

3.4.4 Cylinders

Cylinder components are described in the spacecraft description file with the CYL_X header. Cylinders have a circular cross section with open ends and, consequently, must be capped with planar circle components at each end in order to specify a closed cylindrical object. The surface normal of a cylinder is assumed to be pointing outwards, perpendicularly away from the cylinder axis (see Figure 7b).

The dimensions of a cylinder component are recorded in the spacecraft description file by specifying the radius of the cross-section of the cylinder, followed by the positions of the end-points of the cylinder axis (see Figure 7a).

3.4.5 Spheres

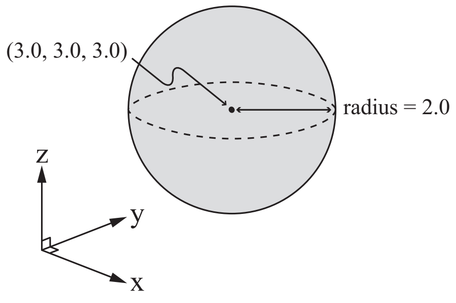
Spherical components are described using the SPHERE specifier in the component header of a spacecraft description file. The radius of the sphere is given on the line after the comment line, followed by the position of the centre of the sphere on the next line. This is shown in Figure 8.

3.4.6 Paraboloids

'Inward' or 'outward' pointing paraboloid components are specified in the spacecraft description file using PARABI or PARABO in the component header, respectively. Paraboloids have a circular open end. Their geometry is recorded in the description file by specifying their depth (the distance between the apex and the base) after the comment line. Then, on the next line, the radius of the open end of the paraboloid

```
SPHERE
8 // 000 0001 shiny metal ball
2.0
3.0 3.0 3.0
0.9 0.9
```

(a)

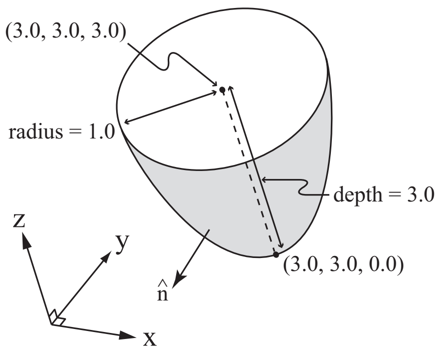


(b)

Figure 8: A sphere component. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a diagram of the component.

```
PARABO
9 // 000 0001 paraboloid surface
3.0
1.0
3.0 3.0 3.0
3.0 3.0 0.0
0.4 0.1
```

(a)



(b)

Figure 9: An 'outward' pointing paraboloid component. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a diagram of the component. The vector $\hat{n}$ points in the direction of the surface normal.

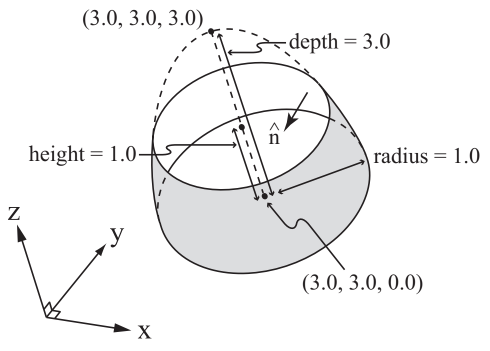
is given. Finally, the position of the centre of the open end, followed by the position of the apex is given. An example of an 'outward' pointing paraboloid is shown in Figure 9. An 'inward' pointing paraboloid would look identical but the surface of consideration would then be the inner (concave) hull.

3.4.7 Truncated paraboloids

'Inward' or 'outward' pointing truncated paraboloid components are specified in the spacecraft description file using a `TPARABI` or `TPARABO` specifier in the component header, respectively. Similar to paraboloid components, truncated paraboloid components have a circular open end, but also have a circular truncated end. They are described in the spacecraft description file by specifying the depth of the paraboloid on the line following the comment line; the radius of the open end on the next line; and, the height (the vertical distance between the open end and the truncated end) on the line after. Finally, the position of the centre of the open end, followed by the

```
TPARABI
10 // 000 0001 truncated paraboloid
3.0
1.0
3.0 3.0 3.0
3.0 3.0 0.0
0.4 0.1
```

(a)

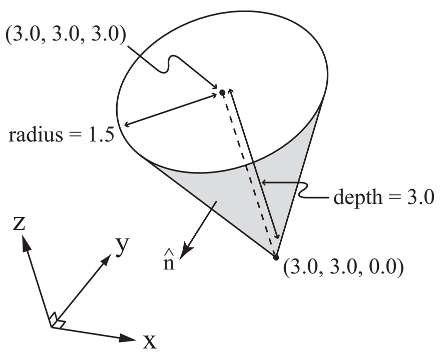


(b)

Figure 10: An 'inward' pointing truncated paraboloid component. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a diagram of the component. The vector $\hat{n}$ points in the direction of the surface normal.

```
CONEO
11 // 000 0001 ice cream cone
3.0
1.5
3.0 3.0 3.0
3.0 3.0 0.0
0.4 0.1
```

(a)



(b)

Figure 11: An 'outward' pointing cone component. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a diagram of the component. The vector $\hat{n}$ points in the direction of the surface normal.

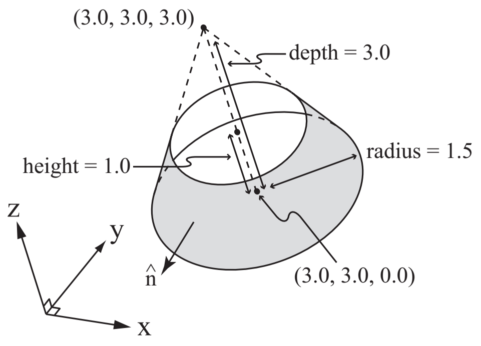
position of the apex of the paraboloid is given. This is illustrated in Figure 10.

3.4.8 Cones

A cone shaped component is assumed to have an open circular cross-section (see Figure 11). It may be specified in the spacecraft description file by using the `CONEI` or `CONEO` keyword in the component header, depending on whether the surface normal of the cone is pointing 'inwards' or 'outwards', respectively. The geometry of the cone is specified by giving its depth on the line after the comment line. The radius of the open end of the cone is given on the next line. The position of the open end of the cone, and the position of the apex are given on the following lines.

```
TCONEO
12 // 000 0001 truncated cone of MLI
3.0
1.5
1.0
3.0 3.0 0.0
3.0 3.0 3.0
0.1 0.5
```

(a)



(b)

Figure 12: An 'outward' pointing truncated cone component. **(a)** shows a representation of the component in the spacecraft description file. **(b)** shows a diagram of the component. The vector $\hat{n}$ points in the direction of the surface normal.

3.4.9 Truncated cones

The truncated cone component is specified in the spacecraft description file with `TCONEI` or `TCONEO` in the record header, representing an 'inward' pointing truncated cone or an 'outward' pointing truncated cone, respectively. As with the cone component, a truncated cone is assumed to have a circular open end. After the comment line, the depth of the truncated cone (distance between open end and apex) is given. The radius of the open end of the component is specified on the next line, followed by the vertical height of the component on the line after. The position of the centre of the open end and the position of the apex are given on the following lines. An example of an 'outward' pointing truncated cone component is shown in Figure 12.

3.5 The output file

The results of the analytical SRP and TRR software are output to the file specified as the third argument in the command line (see subsection 3.1). The file is formatted as a comma separated value (CSV) file.

Each output file begins with a header line that describes each of the values in a line. Thereafter the actual results are given, with one line of results per pixel array orientation. Each line consists of: the latitude of the Sun in the BFS reference frame in degrees; the longitude of the Sun in the BFS reference frame in degrees; the accelerations due to SRP and/or TRR along the BFS x-axis, y-axis and z-axis, respectively, in units of ms^{-2}; and, the EPS angle of the Sun in degrees. If the area profiling option is chosen (see Section 1.1) the accelerations are replaced with the area profile (in units of m^2) of the spacecraft.

3.6 Checkpointing

This version of the analytical SRP and TRR software includes a checkpointing feature which ensures that the results of a run are not lost should the software terminate unexpectedly (for example, by having its process thread terminated). This feature immediately writes results from any process/node to a file with the same path and name as the output file but with a "`.chk`" suffix. This file is only deleted once the output file has been successfully written by the software.

The checkpoint file is a space delimited ASCII file, with each line corresponding to the results obtained from a specific orientation of the pixel array. A header line describes each of the values in a line. Each line consists of: the processor/node number returning the result; the index of the orientation scheme used in the run; the latitude and longitude of the Sun in the BFS reference frame in degrees; the EPS angle of the Sun in degrees; and finally, the accelerations due to SRP and/or TRR along the BFS x-axis, y-axis and z-axis, respectively, in units of ms^{-2}. If the area profiling option is chosen (see Section 1.1) the accelerations are replaced with the area profile (in units of m^2) of the spacecraft.

4 Summary

This document has described the analytical SRP and TRR modelling software developed by members of the Department of Geomatic Engineering [3] at University College London. In particular, the practical aspects of the software (i.e. configuring, compiling and running) have been described for both the Windows and UNIX operating platforms. Readers are referred to the references herein for the theoretical aspects of the software.

The production of this version of the analytical SRP and TRR modelling software and this associated manual was made possible through a grant (grant number GR/T18608/01) from the Engineering and Physical Sciences Research Council (EPSRC) [4] to whom the authors are grateful. We would also like to thank the UCL Research Computing services for the use of their high performance Altix resource [15].

The following appendices provide more information about specific issues with the software, including how to compile MPI applications in Microsoft Visual Studio, and how to execute the analytical SRP and TRR modelling software on high performance computing resources maintained at UCL.

A Tables

Table 1: Compilation options for analytical SRP and TRR modelling software

Definition	**Description**
VERSION	Version number of software
TINY	Constant used in numerical stability tests
M_PI	The value of Pi (π)
M_PI_2	$\pi/2$
M_PI_4	$\pi/4$
D2R	Conversion factor for degrees to radians ($\pi/180$)
R2D	Conversion factor for radians to degrees ($180/\pi$)
MAX_CHARS	Maximum number of alphanumeric characters in a string
SOL_IRR	Solar irradiance [$\mathrm{Wm^{-2}}$]
LIGHT	Speed of light in vacuo [$\mathrm{ms^{-1}}$]
EPSILON	Emissivity of MLI
EPSILON_EFF	Effective emissivity between MLI and spacecraft interior
SIGMA	Stefan-Boltzmann constant [$\mathrm{Wm^{-2}K^{-4}}$]
T_SC	Temperature of spacecraft bus interior [K]
BUFFER	Buffer distance added to limits of pixel array [m]
CARTESIANS_IN_POLY	Maximum number of vectors used to describe a polygon
MAX_COMPS	Maximum number of components in a spacecraft
USE_BLOCK_MODELLING	Flag to invoke block modelling routines - see Section 1.2 (0 = off, 1 = on)
VSMALL	Minimum value used in block modelling routines
NOT_AS_VSMALL	Value used in block modelling tolerances for hit detector
DIST	Distance between spacecraft origin (in body-fixed frame) and pixel array [m]
SP_CONST	Constant used in sphere points algorithm - see Section 3.2.2
USE_MPI	Flag to invoke use of MPI parallelisation - see Section 1.3 (0 = off, 1 = on)
MAX_NODE	Maximum number of nodes allowed during MPI execution
DEBUG_MODE	Flag to invoke the printing of debug statements (0 = off, 1 = on)

Table 2: Parameter keywords for analytical SRP and TRR modelling software

Keyword	**Value type**	**Description**
`model_type`	0, 1, 2 or 3	Type of non-conservative force to model: 0 : SRP only 1 : SRP and TRR 2 : TRR only 3 : Area profile
`scheme`	0 or 1	Type of pixel array orientation scheme to use: 0 : EPS angle range (section 3.2.1) 1 : Sphere points algorithm (section 3.2.2)
`eps_start`	double	Start of EPS angle range† [degrees]
`eps_finish`	double	End of EPS angle range† [degrees]
`eps_delta`	double	EPS angle increment† [degrees]
`k_start`	integer	Start of sphere points index range‡
`k_finish`	integer	End of sphere points index range‡
`n_points`	integer	Total number of sphere points orientations to compute‡
`spacing`	double	Resolution of pixel array [m]
`sr_option`	Y or N	Flag for secondary reflections
`perturbed`	Y or N	Flag for perturbed EPS angle range†
`perturbation`	double	EPS angle scheme perturbation angle† [degrees]
`mass`	double	Mass of spacecraft [kg]
`emissivity`	double	Effective emissivity of MLI for TRR modelling

† = required by EPS angle range pixel array orientation scheme

‡ = required by sphere points algorithm pixel array orientation scheme

B Software history

The following table gives a brief version and edit history of the analytical SRP and TRR modelling software.

Version	Date	Author	Notes
	10 Apr 2002	MKZ	Additions for new component types: rings, paraboloids and conics
	10 Apr 2002	MKZ	Ring testing completed
	14 May 2002	MKZ	Paraboloid testing completed
2.02	21 May 2003	MKZ	Additional components: sphere
2.03	14 Aug 2003	MKZ	Additional components: cone, inward and outward pointing normals
2.04	15 Aug 2003	MKZ	Additional components: truncated cones and paraboloids, inward and outward pointing normals
3.00	27 Aug 2003	MKZ	Combined thermal and SRP modelling
3.02	20 Oct 2003	MKZ	Added option to specify MLI emissivity. Added option to compute model at higher than an EPS angle step size of 5°(introduced to accommodate USAF contract to analyse IIR model at 2°step size)
4.00	27 Nov 2003	AS	Added block modelling option to increase computational efficiency.
4.01	29 Jan 2004	MKZ	altered `write_data_file` to also output BFS lat and lon of Sun position.
4.02	9 Feb 2004	AS	Altered pixel array algorithms for easier perturbed array generation. `get_limits` and `get_perturbed_limits` were reduced to `get_limits2` which can project the spacecraft onto any array no matter what its orientation. The pixel array loop control was altered to handle the generation of a rectangular array, and rather than being generated in a fixed frame and then rotated out to the correct orientation, the individual pixels are produced by a vector sum of pixel sized increments along array local X and Y axes which reduces the number of computations required for each pixel from 9 products and 9 sums, to just 6 sums – should decrease run time.

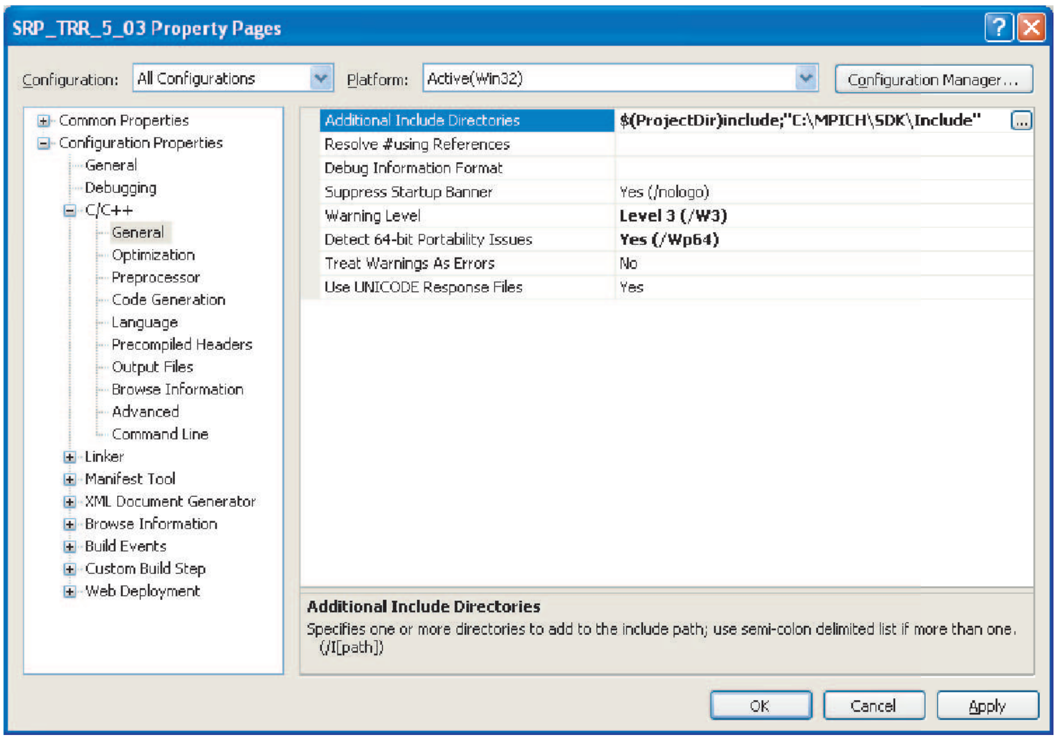
Version	Date	Author	Notes
4.03	23 May 2005	MYG	Code split into different `.CPP` and `.H` files according to function/object. Allows for easier maintenance and debugging, as well as decrease in compilation time. Input parameters read in through parameter file rather than at stdin. MPI calls added to parallelise `SRP_TRR` algorithm across EPS angles. `definitions.h` header file added to control aspects of compilation. Block modelling and MPI usage is now controlled from this header file, as are physical and numerical constants. Spacecraft object altered to be able to read in spacecraft userfile. `FourierTable` object introduced to record accelerations as well as write them to file. `PixelArray` object introduced to simulate position of pixel array in spherical coordinates (r, lat, lon). No longer require `get_limits2` or any of the perturbation functions. Various changes to existing code in order to increase performance (e.g. change of ordering of if–else statements etc.)
5.00	25 Aug 2005	MYG	Software altered to calculate accelerations for orientations of the pixel array distributed equidistance on a sphere in body-fixed frame. Required command line arguments, format of parameter file and format of output data have also been changed.
5.01	19 Oct 2005	MYG	Alterations to `ParameterTable` object so that sphere points algorithm requires user to specify attributes of algorithm (i.e. `n_points`, `k_start`, `k_end`) in parameter file instead of geodesic distance between points on a sphere of 1m radius. Other objects and functions have been changed to reflect these alterations. Thus can now use sphere points pixel array orientation scheme for subrange of k values.
5.02	20 Nov 2005	AS	Alterations to component structure and general/circle/ring intersection routines so that required transformations are pre-computed once when the userfile is first read, rather than computing them each time for every pixel. This is purely intended to reduce runtime somewhat.

Version	Date	Author	Notes
5.03	26 Jun 2006	MYG	Added ability to output area profile of spacecraft for each orientation of the pixel array. This feature is selected by setting the `model_type` option to 3 in the parameter file. It was added to allow spacecraft area profiles to be determined for atmospheric drag calculations. Some bugs also fixed.
5.04	11 Sep 2006	MYG	Checkpointing mechanisms added to the software. This feature immediately writes results to a file, that is eventually deleted upon completion. This ensures that the results are available should the software terminate before completion e.g. by having its process thread terminated.
5.05	29 Sep 2006	MYG	Several bug fixes, including fixing a major bug in PixelArray class which might have incorrectly sized and oriented the array in the BFS frame.

C Compiling Microsoft Visual Studio applications with MPI

It can often be useful to use MPI on a computer hosting the Microsoft Windows operating system in order to execute parallelised software. Particularly because if the computer has a single processor the software will run as if it was running on a multi-processor computer, thus allowing for the software to be tested, debugged and configured in the Visual Studio environment. The following instructions describe how to set the properties of a Microsoft Visual Studio 2005 project in order to compile binary executables with MPI.

1. Download and install the MPICH 1 software library for Microsoft Windows from `http://www-unix.mcs.anl.gov/mpi/mpich1/mpich-nt/`. The necessary file is `mpich.nt.1.2.5.exe`.
2. Open your project in Visual Studio and open the project properties dialogue box (choose *Project* → *Properties...* in the menu or press Alt-F7).
3. Set the configuration to *All configurations*, then add the path to the MPI `Include` directory in the *Additional Include Directories* option of the *C/C++* → *General* branch,

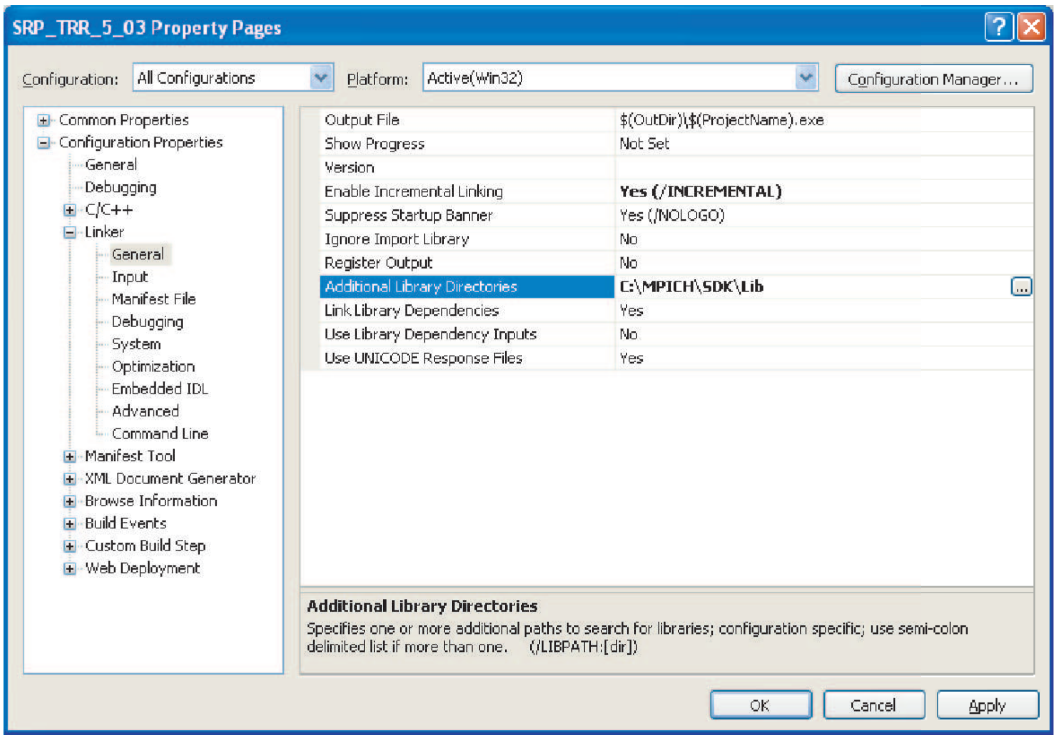


Configuration: All Configurations Platform: Active(Win32)

Additional Include Directories	$(ProjectDir)include;"C:\MPICH\SDK\Include"
Resolve #using References	
Debug Information Format	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	Yes (/Wp64)
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

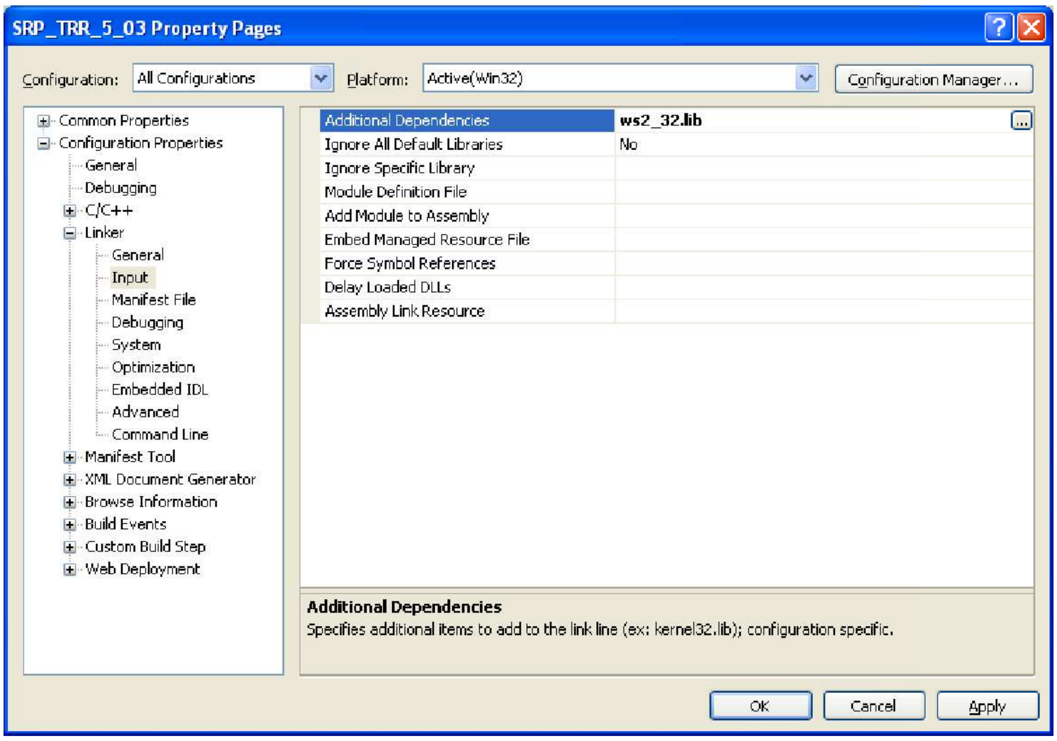
4. In the *Linker* → *General* branch add the path to the MPI `Lib` directory in the *Additional Library Directories* option,



Output File	$(OutDir)\$(ProjectName).exe
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Additional Library Directories	**C:\MPICH\SDK\Lib**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Use UNICODE Response Files	Yes

Additional Library Directories
Specifies one or more additional paths to search for libraries; configuration specific; use semi-colon delimited list if more than one. (/LIBPATH:[dir])

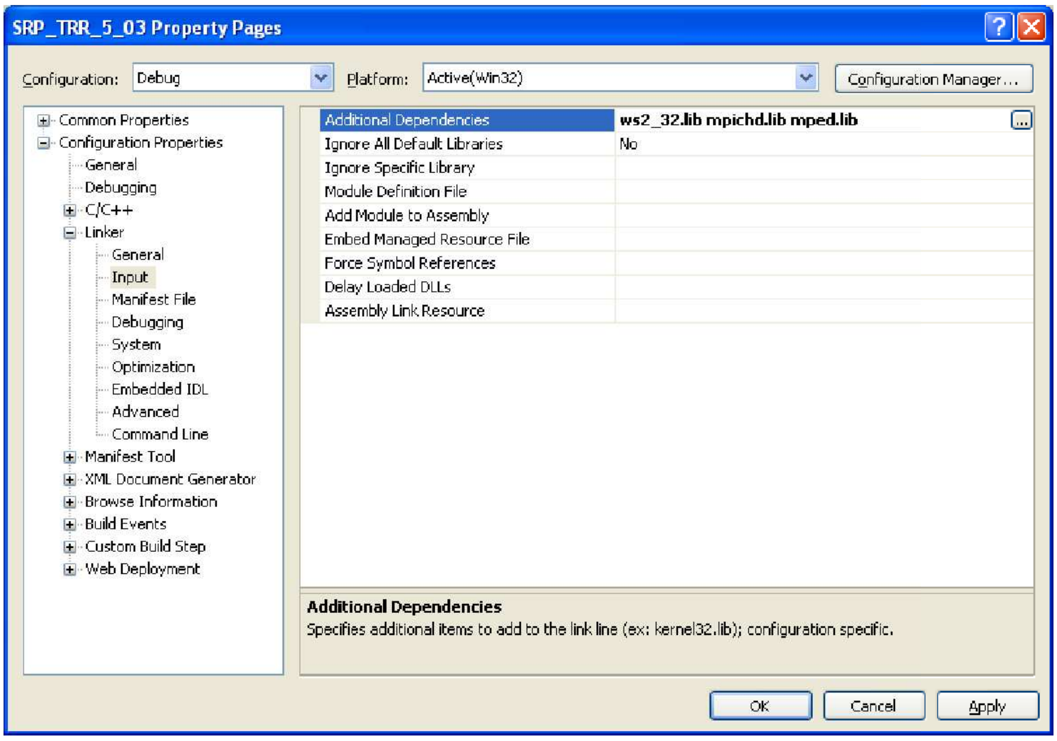
5. In the *Linker* → *Input* branch add `ws2_32.lib` to the list of *Additional Dependencies*, then press the *Apply* button,



Additional Dependencies	**ws2_32.lib**
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

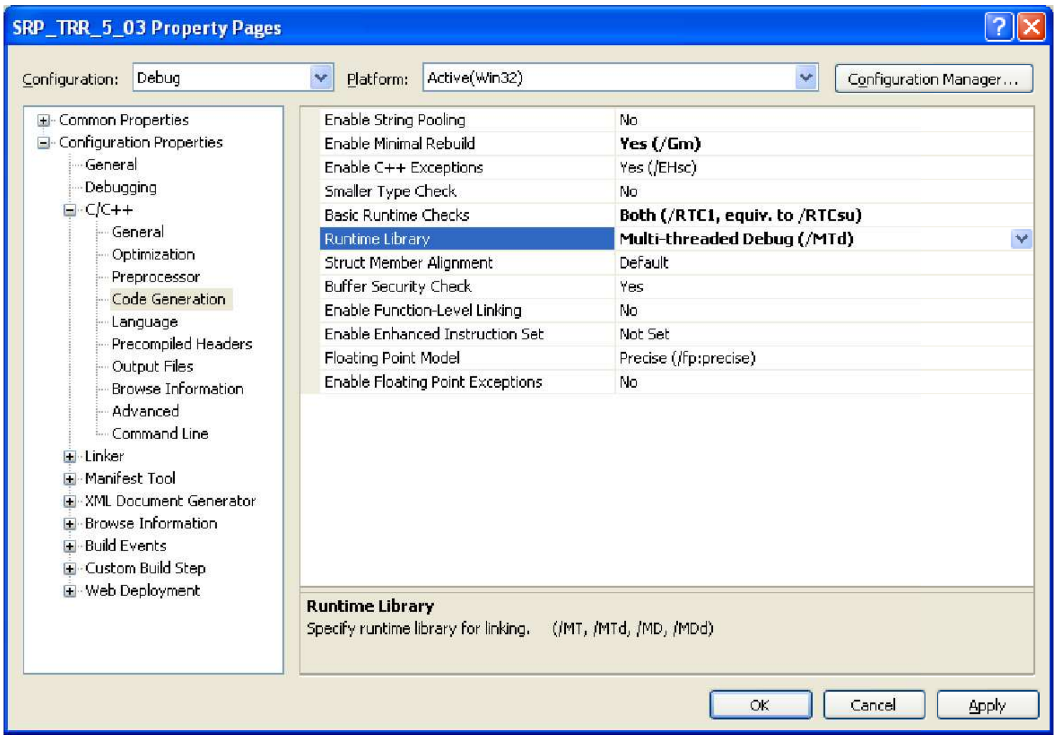
6. Set the configuration to *Debug*, then in the *Linker* → *Input* branch add `mpichd.lib` and `mped.lib` to the list of *Additional Dependencies*,



Additional Dependencies	ws2_32.lib mpichd.lib mped.lib
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

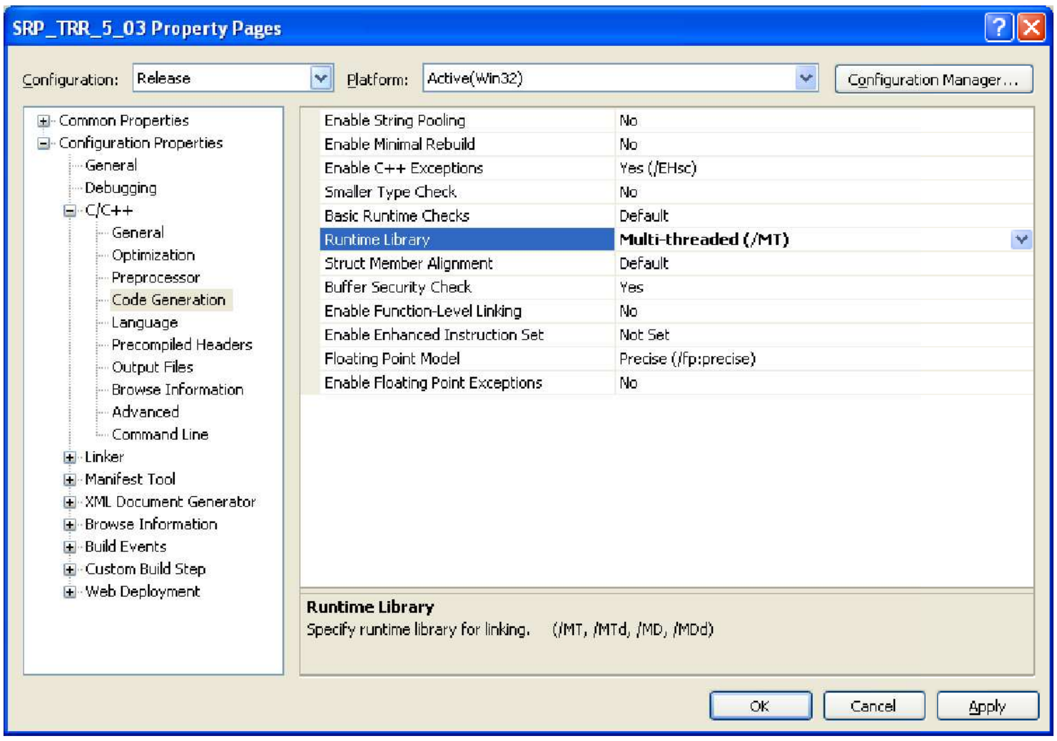
7. In the *C/C++* → *Code Generation* branch change the *Runtime Library* to *Multi-threaded Debug* and press the *Apply* button,



Enable String Pooling	No
Enable Minimal Rebuild	Yes (/Gm)
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu)
Runtime Library	Multi-threaded Debug (/MTd)
Struct Member Alignment	Default
Buffer Security Check	Yes
Enable Function-Level Linking	No
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	No

Runtime Library
Specify runtime library for linking. (/MT, /MTd, /MD, /MDd)

8. Set the configuration to *Release*, then set the *Runtime Library* to *Multi-threaded* in the *Runtime Library* option of the *C/C++* → *Code Generation* branch,

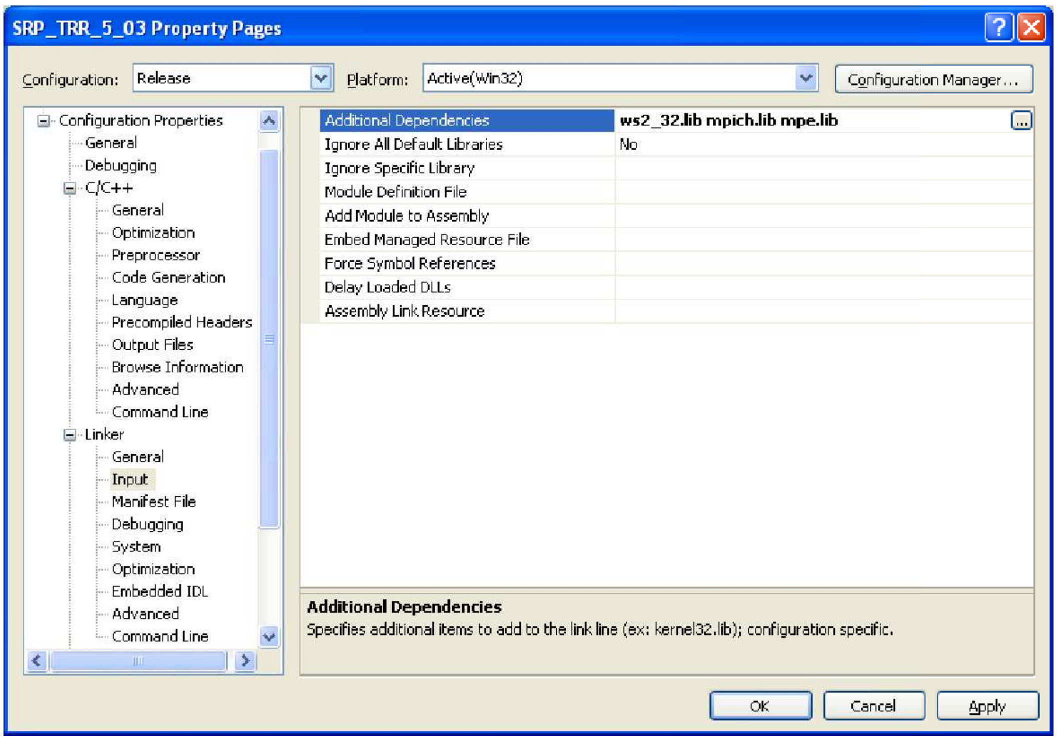


Configuration: Release   Platform: Active(Win32)

Enable String Pooling	No
Enable Minimal Rebuild	No
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	Default
Runtime Library	**Multi-threaded (/MT)**
Struct Member Alignment	Default
Buffer Security Check	Yes
Enable Function-Level Linking	No
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	No

Runtime Library
Specify runtime library for linking.   (/MT, /MTd, /MD, /MDd)

9. In the *Linker* → *Input* branch add `mpich.lib` and `mpe.lib` to the list of *Additional Dependencies*, and press the *OK* button to exit the project properties dialogue.



Configuration: Release   Platform: Active(Win32)

Additional Dependencies	**ws2_32.lib mpich.lib mpe.lib**
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

D Execution on high performance computing resources at UCL

In this appendix we describe how the analytical SRP and TRR modelling software may be compiled, configured and run on high performance computing resources maintained at UCL. In particular, we consider the case where the software is to be executed on the UCL Altix [15]. This is a parallel computing facility managed by UCL Information Systems (IS) and features 56× Itanium2 1.3 Ghz/3 MB cache processors and 112 GB shared memory, offering speeds of upto $\sim$ 135 GFlops.

Access to the UCL Altix is via an SSH terminal such as PuTTY [13] and requires a username and password. This may be requested from the UCL Computing Resource Allocation Group (CRAG) by sending an application email to their members. Full details about the application process may be found at the following URL:
`http://www.ucl.ac.uk/research-computing/community/crag/crag.html#50`

Once access to the UCL Altix has been granted, files may be copied to the resource using secure FTP (SFTP) or secure copy (SCP). Tools such as FileZilla [5] are useful for this purpose.

D.1 Installing the software

After copying the analytical SRP and TRR modelling software to the UCL Altix, it is recommended that the `dos2unix` command is run on the files in the `src` and `include` directories, as well as the `Makefile` to ensure they are compatible with UNIX.

To ensure that the software makes the most efficient use of the UCL Altix resource, the macros in the `include/definitions.h` file should be appropriately set. In particular, the `USE_MPI` macro should be set to `1` and the `MAX_NODE` macro should be set to `56`. See Section 2.3 for more details.

The software may then be compiled by following the instructions in Section 2.5. At the time of writing, the appropriate values for some of the macros in the `Makefile` are as follows:

Macro	Value
`MPIHOME`	`/opt/modules/mpt.1.11-100`
`CC`	`icc`
`INCLUDE`	`-I${PWD}/include -I${MPIHOME}/include`
`LIBS`	`-L${MPIHOME}/lib -lm -lmpi`
`FLAGS`	`-O3 -ipo -ansi -mp -w ${INCLUDE} ${LIBS} ${DEFS}`

D.2 Running the software

The UCL Altix uses LSF from Platform Computing [12] as a resource allocation and queuing system. Short jobs (< 15 minutes) may be executed interactively within the SSH terminal window, but all longer jobs (> 15 minutes) must be submitted and run through LSF. The LSF `bsub` (batch submission) command is used to submit all jobs. This may be as simple as submitting the job at the command line, e.g.

```
bsub -n 8 pam -mpi -auto_place full_path_to_executable/executable
```

A more appropriate way, however, is to create a UNIX Shell script that describes the job that is to be executed. Such a script may then be submitted to the queuing system with a redirection, e.g.

```
bsub < submission_script
```

Each `bsub` command line option is stated on a new line in a submission script, and begins with the text `#BSUB`. This is followed by the option specifier and, where necessary, values for that option. More details about the possible command line options may be found in the manual pages for the `bsub` command (i.e. `man bsub`).

Figure 13 shows an example LSF submission script for the analytical SRP and TRR software on the UCL Altix. The first line defines the script as a C-Shell (CSH) script. The next six lines are `bsub` options as follows:

`-W` Sets the maximum (wall clock) time the job is expected to take.

`-N` Sets the batch queuing system to send an email to the user when the job has completed running.

`-u` Sets the email address to which all emails are to be sent.

`-J` Sets the name of the job.

`-o` Sets the name of the file to which any `stdout` will be piped.

`-n` Sets the number of processors to use.

The last five lines of Figure 13 state the actual command to be executed by the queuing system. Here, the parallel application manager (`pam`) is used to submit the job as an MPI parallelised application, ensuring that the software will run over several processors. More details about this command may be found in the corresponding manual pages (i.e. `man pam`). The analytical SRP and TRR modelling software is called as described in Section 3.1.

```
#!/bin/csh
#BSUB -W 18:00
#BSUB -N
#BSUB -u user@ge.ucl.ac.uk
#BSUB -J my_job
#BSUB -o /home/disk5/user/my_job/stdout.txt
#BSUB -n 16
pam -mpi -auto_place \
/home/disk5/user/srp_trr_5_05/bin/srp_trr_5_05 \
/home/disk5/user/my_job/parameters.txt \
/home/disk5/user/my_job/userfile.txt \
/home/disk5/user/my_job/results.csv
```

Figure 13: An example of a LSF submission script for the UCL Altix. See text for details.

D.3 Managing jobs

To view the list of your jobs submitted to the UCL Altix, and their status (i.e. running or waiting to be run) use the `qstat` command. The `-a` option with this command allows one to see the status of all jobs on the system.

To remove a job from the system, whilst it is running or whilst it is waiting to be run, use the `qdel` command followed by request ID number of the job. This can be found by using the `qstat` command.

Further details about both the commands above may be found on their corresponding manual pages.

E Execution on HPCx

Here we describe how the analytical SRP and TRR modelling software may be compiled, configured and run on the HPCx national supercomputing facility [8]. HPCx consists of several IBM eServer 575 nodes, featuring a total of 1536 $\times$ 1.5 Ghz IBM Power5 processors, with 32 GB memory per frame of 16 processors. This offers a theoretical peak of $\sim$ 7395 GFlops, putting the facility at number 59 in the Top500 list [17] for July 2006.

Access to the HPCx facilities is via an SSH terminal such as PuTTY [13] and requires a username and password. This may be requested through the HPCx website at: `https://www.hpcx.ac.uk/signup.jsp` The “Project Code” and “Project Password” may be acquired from the Project Manager or Principle Investigator (PI) managing the project.

Files must be copied to/from the HPCx servers using SFTP or SCP.

E.1 Installing the software

Before copying the analytical SRP and TRR modelling software to the HPCx servers it is recommended that the files in the `src` and `include` directories, as well as the `Makefile`, are converted to UNIX format, possibly by using a utility such as `dos2unix` within Cygwin [2].

To ensure that the software makes the most efficient use of the HPCx resources, the macros in the `include/definitions.h` file should be appropriately set. In particular, the `USE_MPI` macro should be set to `1` and the `MAX_NODE` macro should be set to `1024`. See Section 2.3 for more details.

The software may then be compiled by following the instructions in Section 2.5. At the time of writing, the appropriate values for some of the macros in the `Makefile` are as follows:

Macro	Value
`MPIHOME`	*leave blank*
`CC`	`mpCC_r`
`INCLUDE`	`-I${PWD}/include`
`LIBS`	`-lm`
`FLAGS`	`-q64 -O3 -qarch=pwr4 -qtune=pwr4 ${INCLUDE} ${LIBS} ${DEFS}`

E.2 Running the software

The HPCx system uses IBM's LoadLeveler to build, submit and process batch jobs. The following runtime limits have been imposed on the system:

Max. processors	Max. runtime
Serial	12 hours
16	6 hours
64	6 hours
128	12 hours
1024	12 hours

All batch jobs should use multiples of 16 processors. If the number of processors requested is not a multiple of 16, users will be charged the next largest multiple of 16 processors.

Each job to the LoadLeveler must be submitted using a job command file, which describes the job to be submitted and contains a number of LoadLeveler keyword statements which specify the various resources needed by the job. A comprehensive description of job command files for various types of jobs may be found in the HPCx user guide [9].

A job may be submitted to the LoadLeveler using the `llsubmit` command i.e.

```
llsubmit job_command_file
```

Upon submission the LoadLeveler performs a number of checks on the job command file. If any problems are found an error message is returned and the job is not submitted to the batch system.

E.3 Managing jobs

To view the list of jobs submitted to the HPCx system, and their status (i.e. running or waiting to be run) use the `llq` command. There are usually many jobs on the system at any one time so it is often useful to `grep` the results of the `llq` command to view specific jobs only. For example,

```
llq | grep joe_bloggs
```

will only show the status of jobs submitted by user `joe_bloggs`.

To remove a job from the system, whilst it is running or whilst it is waiting to be run, use the `llcancel` command followed by the request ID number of the job. This can be found by using the `llq` command.

Other useful commands and their usage may be found in the corresponding pages of the HPCx user guide [9].

References

[1] Adhya, S. (2005) Thermal re-radiation modelling for the precise prediction and determination of spacecraft orbits, Ph.D. Thesis, Univ. of London, London, U.K.

[2] Cygwin: `http://www.cygwin.com/`

[3] Department of Geomatic Engineering, University College London: `http://www.ge.ucl.ac.uk/`

[4] Engineering and Physical Sciences Research Council: `http://www.epsrc.ac.uk/`

[5] FileZilla: `http://filezilla.sourceforge.net/`

[6] GNU Make: `http://www.gnu.org/software/make/`

[7] Gropp, W., E. Lusk and A. Skjellum (1999) Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, Massachusetts, U.S.A.

[8] HPCx: `http://www.hpcx.ac.uk/`

[9] HPCx User Guide: `http://www.hpcx.ac.uk/support/documentation/UserGuide/HPCxuser/HPCxuser.html`

[10] Microsoft Visual Studio: `http://msdn.microsoft.com/vstudio/`

[11] mpirun: `http://www-unix.mcs.anl.gov/mpi/www/www1/mpirun.html`

[12] Platform Computing: `http://www.platform.com/`

[13] PuTTY: `http://www.chiark.greenend.org.uk/~sgtatham/putty/`

[14] Saff, E.B., and A.B.J. Kuijlaars (1997) Distributing Many Points on a Sphere, *The Mathematical Intelligencer*, **19**, No. 1, pp. 5-11.

[15] SGI Altix 3700 at UCL: `http://www.ucl.ac.uk/research-computing/services/altix/altix.html`

[16] Sibthorpe, A. (2006) Precision non-conservative force modelling for low earth orbiting spacecraft, Ph.D. Thesis, Univ. of London, London, U.K.

[17] Top500: `http://www.top500.org/`

[18] Ziebart, M.K., S. Adhya, A. Sibthorpe and P. Cross (2003) GPS Block IIR non-conservative force modelling: Computation and implications, in *Proceedings of ION GPS/GNSS 2003*, Portland, Oregon, U.S.A.

[19] Ziebart, M.K. (2004) Generalised analytical solar radiation pressure modelling algorithm for spacecraft of complex shape, *Journal of Spacecraft and Rockets*, **41**, No. 5, pp. 840-848.