5.4 Astronomical Images

5.4.1 Overview

Above, we have defined a basic container for a single 2D collection of pixels (`psImage`), along with basic operations to manipulate the image pixels. For astronomical applications, this data structure is insufficient for two reasons. First, it does provide sufficient additional metadata to describe the data in detail. Second, astronomy applications frequent involve multiple, related images. For PanSTARRS, and for general astronomical applications, we require a richer collection of data structures which describe a very general image concept. We have defined several layers in the hierarchy which are necessary to describe the image data which will be produced by the PanSTARRS Gigapixel cameras as well as other standard astronomical images.

A simple 2D image is a basic data unit for much of astronomical imaging. If we consider various optical and IR array cameras, a single readout of the detector produces a collection of pixels measurements. We define our lowest-level astronomical image structure, `psReadout`, to contain the pixels produced by a single readout of the detector, along with metadata needed to define that readout: the origin and binning of the image relative to the original detector pixels explicitly in the structure, and pointers to the general metadata and derived objects, if any.

A single detector may produce more than one read which is associated. For example, infrared detectors frequently produce an image immediately after the detector is reset followed by an image after the basic exposure is complete. Both readouts correspond to the same pixels, though the binning or rastering may be different between the two readouts. Another example is the video sequence produced by the PanSTARRS Gigapix camera guide cells, each of which represents a series of many images from a subraster of pixels in the detector readout portion. The second level of our image container hierarchy, `psCell`, consists of a collection of readouts from a single detector.

In the PanSTARRS Gigapix camera, the basic readout region is a fraction of the full imaging area of a single CCD chip. The chip is divided into 64 cells, any fraction of which may have been readout for a given exposure. In other cameras, such as Megacam at CFHT, the individual CCDs have multiple amplifiers addressing contiguous portions of the detector. In such cameras, each amplifier produces a separate collection of pixels. In the third level of our image container hierarchy, the data structure `psChip` represents a collection of different cells.

The top level of our image container hierarchy is a complete focal plane array (`psFPA`). This structure represents the collection of chips in the camera, all of which are read out in a given exposure.

For example, take a mosaic camera consisting of eight $2k \times 4k$ CCDs, each of which is read out through two amplifiers. Then there would be sixteen cells in total, each of which is presumably $2k \times 2k$. There would be eight chips, each consisting of two cells, and the focal plane consists of these eight chips.

As another example, consider an observation by PS1. The focal plane would consist of 60 chips, each of which consist of 64 cells (or less; a few cells may be dead). Some cells (those containing guide stars for the orthogonal transfer) will contain multiple readouts.

These data structures represent containers with which to carry around the collection of related image data. There is no requirement on the functions or the structures that each instance of one of these data structures represent the physical hardware. For example, it is not necessary that an instance of `psFPA` always carry the data for all

60 (or 64) Gigapixel camera OTAs. The usage of these structures is such that all astronomical operations which apply to a CCD image should be performed on an instance of `psFPA`. If a particular circumstance only requires a single 2D image, then that is represented by an instance of `psFPA` with one `psChip`, which in turn has one `psCell`, which in turn has one `psReadout`.

These container levels also include in their definition the information needed to transform the coordinates in one of the levels to the coordinate system relevant at the higher levels.

5.4.2 A Readout

A readout is the result of a single read of a cell (or a portion thereof). It contains a pointer to the pixel data, and additional pointers to the objects found in the readout, and the readout metadata. It also contains the offset from the lower-left corner of the chip, in the case that the CCD was windowed.

```
typedef struct {
    const int x0, y0; ///< Offset from the lower-left corner of the physical device
    const int xBin, yBin; ///< Image binning
    psImage *image;                    ///< imaging area of cell
    psDlist *objects; ///< objects derived from cell
    psMetadata *md; ///< Readout-level metadata
} psReadout;
```

5.4.3 A Cell

A cell consists of one or more readouts (usually only one except in the case that the cell has been used for fast guiding). It also contains a pointer to the cell metadata, and a pointer to its parent chip. On the astrometry side, it also contains coordinate transforms from the cell to the chip and, as a convenience, from the cell to the focal plane. It is expected that these transforms will consist of two first-order 2D polynomials, simply specifying a translation, rotation and magnification; hence they are easily inverted, and there is no need to add reverse transformations. We also add an additional transformation, which is intended to provide a "quick and dirty" transform from the cell coordinates to the sky; this transformation not guaranteed to be as precise as the "standard" transformation of Cell → Chip → Focal Plane → Tangent Plane → Sky, but will be faster.

```
typedef struct {
    int nReadouts; ///< number of readouts in this cell
    struct psReadout *readouts; ///< Readouts from the cell
    psMetadata *md; ///< Cell-level metadata
    psPlaneTransform *cellToChip; ///< Transformations from cell coordinates to chip coordinates
    psPlaneTransform *cellToFPA; ///< Transformations from cell coordinates to FPA coordinates
    psPlaneTransform *cellToSky; ///< Quick and Dirty transformations from cell coordinates to sky
    struct psChip  *parentChip; ///< chip which contains this cell
} psCell;
```

5.4.4 A Chip

A chip consists of one or more cells (according to the number of amplifiers on the CCD). It contains a pointer to the chip metadata, and a pointer to the parent focal plane. For astrometry, it contains a coordinate transform from the chip to the focal plane. It is expected that this transforms will consist of two second-order 2D polynomials; hence we think that it is prudent to include a reverse transformation which will be derived from numerically inverting the forward transformation.

```
typedef struct {
    int nCells;                         ///< Number of Cells assigned
    struct psCell *cells; ///< Cells in the Chip
    psMetadata *md; ///< Chip-level metadata
    psPlaneTransform *chipToFPA; ///< Transformations from chip coordinates to FPA coordinates
    psPlaneTransform *FPAtoChip; ///< Transformations from FPA coordinates to chip
    struct psFPA *parentFPA; ///< FPA which contains this chip
} psChip;
```

5.4.5 A Focal Plane

A focal plane consists of one or more chips (according to the number of pieces of contiguous silicon). It contains pointers to the focal plane metadata and the exposure information. For astrometry, it contains a transformation from the focal plane to the tangent plane and the fixed pattern residuals. It is expected that the transformation will consist of two 4D polynomials (i.e. a function of two coordinates in position, the magnitude of the object, and the color of the object) in order to correct for optical distortions and the effects of the atmosphere; hence we think that it is prudent to include a reverse transformation which will be derived from numerically inverting the forward transformation. Since colors are involved in the transformation, it is necessary to specify the color the transformation is defined for. We also include some values to characterize the quality of the transformation: the root mean square deviation for the x and y transformation fits, and the χ^2 for the transformation fit.

```
typedef struct {
    int nChips;                         ///< Number of Cells assigned
    int nAlloc;                         ///< Number of Cells available
    struct psChip *chips; ///< Chips in the Focal Plane Array
    psMetadata *md; ///< FPA-level metadata
    psPlaneDistort *TPtoFP; ///< Transformation term from
    psPlaneDistort *FPtoTP; ///< Transformation term from
    psFixedPattern *pattern; ///< Fixed pattern residual offsets
    const psExposure *exp; ///< information about this exposure
    const psGrommit *grommit; ///< Information for conversion from TP to Sky.
    psPhotSystem colorPlus, colorMinus; ///< Colour reference
    float rmsX, rmsY;                   ///< Dispersion in astrometric solution
    float chi2;                         ///< chi^2 of astrometric solution
} psFPA;
```

5.4.6 Exposure information

We need several quantities from the telescope in order to make a first guess at the astrometric solution. From these quantities, further quantities can be derived and stored for later use.

```
typedef struct {
    const double ra, dec; ///< Telescope boresight
    const double ha; ///< Hour angle
    const double zd; ///< Zenith distance
    const double az; ///< Azimuth
    const double lst; ///< Local Sidereal Time
    const psTime *time; ///< MJD of observation
    const float rotAngle; ///< Rotator position angle
    const float temp; ///< Air temperature, for estimating refraction
    const float pressure; ///< Air pressure, for calculating refraction
    const float humidity; ///< Relative humidity, for calculating refraction
    const float exptime; ///< Exposure time
    const float wavelength; ///< Wavelength of observation
    const psObservatory *observatory; ///< Observatory data
    /* Derived quantities */
    const float posAngle; ///< Position angle
```

```
    const float parallactic; ///< Parallactic angle
    const float airmass; ///< Airmass, calculated from zenith distance
    const float pf; ///< Parallactic factor
    const char *cameraName; ///< name of camera which provided exposure
    const char *telescopeName; ///< name of telescope which provided exposure
} psExposure;
```

5.4.7 Observatory data

We need a container for the observatory data that doesn't change per exposure.

```
typedef struct {
    const char *name; ///< Name of observatory
    const double latitude; ///< Latitude of observatory, east positive
    const double longitude; ///< Longitude of observatory
    const double height; ///< Height of observatory
    const double tlr; ///< Tropospheric Lapse Rate
} psObservatory;
```

5.4.8 Constructors

Each of the above structures needs an appropriate constructor and destructor. In what follows, each of the variables has the same meaning as in the appropriate structure.

The constructor for `psExposure` shall be:

```
psExposure *psExposureAlloc(double ra, double dec, double ha, double zd, double az, double lst,
                            const psTime *time, float rotAngle, float temp, float pressure, float humidity,
    float exptime, float wavelength, const psObservatory *observatory);
```

The constructor for `psObservatory` shall be:

```
psObservatory *psObservatoryAlloc(const char *name, double latitude, double longitude,
                                  double height, double tlr);
```

The constructor for `psFPA` shall be:

```
psFPA *psFPAAlloc(int nAlloc, const psExposure *exp);
```

The constructor shall make an empty `psFPA`, with the `nAlloc` allocated pointers to `psChips` being set to `NULL`; all other pointers in the structure shall be initialized to `NULL`, apart from the `grommit`, which shall be constructed on the basis of the `exp`.

The destructor for `psFPA` shall destroy the related `psGrommit`.

The constructor for `psChip` shall be:

```
psChip *psChipAlloc(int nCells, psFPA *parentFPA);
```

The constructor shall make an empty `psChip`, with the `nCells` allocated pointers to `psCells` being set to `NULL`; all other pointers in the structure shall be initialized to `NULL`.

The constructor for `psCell` shall be:

```
psCell *psCellAlloc(int nReadouts, struct psChip *parentChip);
```

The constructor shall make an empty `psCell`, with the `nReadouts` allocated pointers to `psReadouts` being set to `NULL`; all other pointers in the structure shall be initialized to `NULL`.

The constructor for `psReadout` shall be:

```
psReadout *psReadoutAlloc(int x0, int y0, const psImage *image);
```

All pointers in the structure other than the `image` shall be initialized to `NULL`.

5.5 Astrometry

Astrometry is a basic functionality required for the IPP that will be used repeatedly, both for low-precision (roughly where is my favorite object?) and high-precision (what is the proper motion of this star?). As such, it must be flexible, yet robust. Accordingly, we will wrap the StarLink Astronomy Libraries (SLALib), which has already been developed.

5.5.1 Coordinate frames

There are five coordinate frames that we need to worry about for the purposes of astrometry:

- Cell: (x, y) in pixels — raw coordinates;
- Chip: (X, Y) in pixels — the location on the silicon;
- Focal Plane: (p, q) in microns — the location on the focal plane;
- Tangent Plane: (l, m) in arcsec from the telescope boresight; and
- Sky: (RA,Dec) — ICRS.

The following steps are required to convert from the cell coordinates to the sky:

- Cell $\longleftrightarrow$ Chip: two 2D polynomials, $(X, Y) = f(x, y)$;
- Chip $\longleftrightarrow$ FP: two 2D polynomials, $(p, q) = g(X, Y)$;
- FP $\longleftrightarrow$ TP: two 4D polynomials, $(l, m) = h(p, q, m, c)$, where m and c are the magnitude and color of the object, respectively; and
- TP $\longleftrightarrow$ Sky: SLALib transformation using a transform pre-computed for each pointing.

Note that the transformation between the Focal Plane and the Tangent Plane is a four-dimensional polynomial, in order to account for any possible dependencies in the astrometry on the stellar magnitude and color; the former serves as a check for charge transfer inefficiencies, while the latter will correct chromatic refraction, both through the atmosphere and the corrector lenses.

We require structures to contain each of the above transformations as well as the pixel data.