```
% LFUtilDecodeLytroFolder - decode and optionally colour correct and rectify Lytro light fields
%
% Usage:
%
%     LFUtilDecodeLytroFolder
%     LFUtilDecodeLytroFolder( InputPath )
%     LFUtilDecodeLytroFolder( InputPath, FileOptions, DecodeOptions, RectOptions )
%     LFUtilDecodeLytroFolder( InputPath, [], [], RectOptions )
%
% All parameters are optional and take on default values as set in the "Defaults" section at the top
% of the implementation. As such, this can be called as a function or run directly by editing the
% code. When calling as a function, pass an empty array "[]" to omit a parameter.
%
% As released, the default values are set up to match the naming conventions of LFP Reader v2.0.0.
%
% This function demonstrates decoding and optionally colour-correction and rectification of 2D
% lenslet images into 4D light fields. It recursively crawls through a prescribed folder and its
% subfolders, operating on each light field. It can be used incrementally: previously-decoded light
% fields can be subsequently colour-corected, rectified, or both. Previously-completed tasks will
% not be re-applied. A filename pattern can be provided to work on individual files. All paths and
% naming are configurable.
%
% Decoding and rectification follow the process described in:
%
% [1] D. G. Dansereau, O. Pizarro, and S. B. Williams, "Decoding, calibration and rectification for
% lenslet-based plenoptic cameras," in Computer Vision and Pattern Recognition (CVPR), IEEE
% Conference on. IEEE, Jun 2013.
%
% Decoding requires that an appropriate database of white images be created using
% LFUtilProcessWhiteImages. Rectification similarly requires a calibration database be created using
% LFUtilProcessCalibrations.
%
% To decode a single light field, it is simplest to include a file specification in InputPath (see
% below). It is also possible to call LFLytroDecodeImage directly.
%
% Colour correction employs the metadata associated with each Lytro picture. It also applies
% histogram-based contrast adjustment. It calls the functions LFColourCorrect and LFHistEqualize.
%
% Rectification employs a calibration info file to rectify the light field, correcting for lens
% distortion, making pixels square, and yielding an intrinsics matrix which allows easy conversion
% from a pixel index [i,j,k,l] to a ray [s,t,u,v]. A calibration info file is generated by
% processing a series of checkeboard images, following the calibration procedure described in
% LFToolbox.pdf. A calibration only applies to one specific camera at one specific zoom and focus
% setting, and decoded using one specific lenslet grid model. The tool LFUtilProcessCalibrations is
% used to build a database of rectifications, and LFSelectFromDatabase isused to select a
% calibration appropriate to a given light field.
%
% This function was written to deal with Lytro imagery, but adapting it to operate with other
% lenslet-based cameras should be straightforward. For more information on the decoding process,
% refer to LFDecodeLensletImageSimple, [1], and LFToolbox.pdf.
%
% Some optional parameters are not used or documented at this level -- see each of LFCalRectifyLF,
% LFLytroDecodeImage, LFDecodeLensletImageSimple, and LFColourCorrect for further information.
%
%
% Inputs -- all are optional, see code below for default values :
%
%     InputPath :  Path to folder containing light fields, or to a specific light field, optionally including one or
%     more wildcard filename specifications. In case wildcards are used, this searches sub-folders recursively. See
%     LFFindFilesRecursive.m for more information and examples of how InputPath is interpreted.
%
%     FileOptions : struct controlling file naming and saving
%                .SaveResult : Set to false to perform a "dry run"
%                 .ForceRedo : If true previous results are ignored and decoding starts from scratch
%          .SaveFnamePattern : String defining the pattern used in generating the output filename;
%                              sprintf is used to complete this pattern, such that %s gets replaced
%                              with the base name of the input light field
%          .ThumbFnamePattern : As with SaveFnamePattern, defines the name of the output thumbnail
%                              image
%
%     DecodeOptions : struct controlling the decoding process, see LFDecodeLensletImageSimple for more info
%                    .OptionalTasks : Cell array containing any combination of 'ColourCorrect' and
%                                     'Rectify'; an empty array "{}" means no additional tasks are
%                                     requested; case sensitive
%          .LensletImageFnamePattern : Pattern used to locate input files -- the pattern %s stands in
%                                     for the base filename
%                  .ColourHistThresh : Threshold used by LFHistEqualize in optional colour correction
%            .WhiteImageDatabasePath : Full path to the white images database, as created by
%                                     LFUtilProcessWhiteImages
%                          .DoDehex : Controls whether hexagonal sampling is converted to rectangular, default true
%                        .DoSquareST : Controls whether s,t dimensions are resampled to square pixels, default true
%                     .ResampMethod : 'fast'(default) or 'triangulation'
```

```
%                     .LevelLimits : a two-element vector defining the black and white levels
%                       .Precision : 'single'(default) or 'double'
%
%     RectOptions : struct controlling the optional rectification process
%         .CalibrationDatabaseFname : Full path to the calibration file database, as created by
%                                     LFUtilProcessCalibrations;
%
% Example:
%
%   LFUtilDecodeLytroFolder
%
%     Run from the top level of the 'Samples' folder will decode all the light fields in all the
%     sub-folders, with default settings as set up in the opening section of the code. The
%     calibration database created by LFUtilProcessWhiteImages is expected to be in
%     'Cameras/CaliCalibrationDatabase.mat' by default.
%
%   LFUtilDecodeLytroFolder('Images', [], struct('OptionalTasks', 'ColourCorrect'))
%
%     Run from the top level of the 'Samples' folder will decode and colour correct all light fields in the Images
%     folder and its sub-folders.
%
%   DecodeOptions.OptionalTasks = {'ColourCorrect', 'Rectify'};
%   LFUtilDecodeLytroFolder([], [], DecodeOptions)
%
%     Will perform both colour correction and rectification in the Images folder.
%
%   LFUtilDecodeLytroFolder('Images/Illum/Lorikeet.lfp')
%   LFUtilDecodeLytroFolder('Lorikeet.lfp')
%   LFUtilDecodeLytroFolder({'Images', '*Hiding*', 'Jacaranda*'})
%   LFUtilDecodeLytroFolder('*.raw')
%   LFUtilDecodeLytroFolder({'*0002*', '*0003*'})
%
%     Any of these, run from the top level of the 'Samples' folder, will decode the matching files.  See
%     LFFindFilesRecursive.
%
% See also: LFUtilExtractLFPThumbs, LFUtilProcessWhiteImages, LFUtilProcessCalibrations, LFUtilCalLensletCam,
% LFColourCorrect, LFHistEqualize, LFFindFilesRecursive, LFLytroDecodeImage, LFDecodeLensletImageSimple,
% LFSelectFromDatabase

% Part of LF Toolbox v0.4 released 12-Feb-2015
% Copyright (c) 2013-2015 Donald G. Dansereau

function LFUtilDecodeLytroFolder( InputPath, FileOptions, DecodeOptions, RectOptions )

%---Defaults---
InputPath = LFDefaultVal( 'InputPath', 'Images' );

FileOptions = LFDefaultField('FileOptions', 'SaveResult', true);
FileOptions = LFDefaultField('FileOptions', 'ForceRedo', false);
FileOptions = LFDefaultField('FileOptions', 'SaveFnamePattern', '%s__Decoded.mat');
FileOptions = LFDefaultField('FileOptions', 'ThumbFnamePattern', '%s__Decoded_Thumb.png');

DecodeOptions = LFDefaultField('DecodeOptions', 'OptionalTasks', {}); % 'ColourCorrect', 'Rectify'
DecodeOptions = LFDefaultField('DecodeOptions', 'ColourHistThresh', 0.01);
DecodeOptions = LFDefaultField(...
    'DecodeOptions', 'WhiteImageDatabasePath', fullfile('Cameras','WhiteImageDatabase.mat'));
RectOptions = LFDefaultField(...
    'RectOptions', 'CalibrationDatabaseFname', fullfile('Cameras','CalibrationDatabase.mat'));
% Used to decide if two lenslet grid models are "close enough"... if they're not a warning is raised
RectOptions = LFDefaultField( 'RectOptions', 'MaxGridModelDiff', 1e-5 );

% Massage a single-element OptionalTasks list to behave as a cell array
while( ~iscell(DecodeOptions.OptionalTasks) )
    DecodeOptions.OptionalTasks = {DecodeOptions.OptionalTasks};
end

%---Crawl folder structure locating raw lenslet images---
DefaultFileSpec = {'*.lfr', '*.lfp', '*.LFR', '*.raw'}; % gets overriden below, if a file spec is provided
DefaultPath = 'Images';
[FileList, BasePath] = LFFindFilesRecursive( InputPath, DefaultFileSpec, DefaultPath );

fprintf('Found :¥n');
disp(FileList)

%---Process each raw lenslet file---
% Store options so we can reset them for each file
OrigDecodeOptions = DecodeOptions;
OrigRectOptions = RectOptions;

for( iFile = 1:length(FileList) )
    SaveRequired = false;
```

```
    %---Start from orig options, avoids values bleeding between iterations---
    DecodeOptions = OrigDecodeOptions;
    RectOptions = OrigRectOptions;

    %---Find current / base filename---
    CurFname = FileList{iFile};
    CurFname = fullfile(BasePath, CurFname);

    % Build filename base without extension, auto-remove '__frame' for legacy .raw format
    LFFnameBase = CurFname;
    [~,~,Extension] = fileparts(LFFnameBase);
    LFFnameBase = LFFnameBase(1:end-length(Extension));
    CullIdx = strfind(LFFnameBase, '__frame');
    if( ~isempty(CullIdx) )
        LFFnameBase = LFFnameBase(1:CullIdx-1);
    end

    fprintf('¥n---%s [%d / %d]...¥n', CurFname, iFile, length(FileList));

    %---Decode---
    fprintf('Decoding...¥n');

    % First check if a decoded file already exists
    [SDecoded, FileExists, CompletedTasks, TasksRemaining, SaveFname] = CheckIfExists( ...
        LFFnameBase, DecodeOptions, FileOptions.SaveFnamePattern, FileOptions.ForceRedo );

    if( ~FileExists )
        % No previous result, decode
        [LF, LFMetadata, WhiteImageMetadata, LensletGridModel, DecodeOptions] = ...
            LFLytroDecodeImage( CurFname, DecodeOptions );
        if( isempty(LF) )
            continue;
        end
        fprintf('Decode complete¥n');
        SaveRequired = true;
    elseif( isempty(TasksRemaining) )
        % File exists, and nothing more to do
        continue;
    else
        % File exists and tasks remain: unpack previous decoding results
        [LF, LFMetadata, WhiteImageMetadata, LensletGridModel, DecodeOptions] = LFStruct2Var( ...
            SDecoded, 'LF', 'LFMetadata', 'WhiteImageMetadata', 'LensletGridModel', 'DecodeOptions' );
        clear SDecoded
    end

    %---Display thumbnail---
    Thumb = DispThumb(LF, CurFname, CompletedTasks);

    %---Optionally colour correct---
    if( ismember( 'ColourCorrect', TasksRemaining ) )
        LF = ColourCorrect( LF, LFMetadata, DecodeOptions );
        CompletedTasks = [CompletedTasks, 'ColourCorrect'];
        SaveRequired = true;
        fprintf('Done¥n');

        %---Display thumbnail---
        Thumb = DispThumb(LF, CurFname, CompletedTasks);
    end

    %---Optionally rectify---
    if( ismember( 'Rectify', TasksRemaining ) )
        [LF, RectOptions, Success] = Rectify( LF, LFMetadata, DecodeOptions, RectOptions, LensletGridModel );
        if( Success )
            CompletedTasks = [CompletedTasks, 'Rectify'];
            SaveRequired = true;
        end
        %---Display thumbnail---
        Thumb = DispThumb(LF, CurFname, CompletedTasks);
    end

    %---Check that all tasks are completed---
    UncompletedTaskIdx = find(~ismember(TasksRemaining, CompletedTasks));
    if( ~isempty(UncompletedTaskIdx) )
        UncompletedTasks = [];
        for( i=UncompletedTaskIdx )
            UncompletedTasks = [UncompletedTasks, ' ', TasksRemaining{UncompletedTaskIdx}];
        end
        warning(['Could not complete all tasks requested in DecodeOptions.OptionalTasks: ', UncompletedTasks]);
    end

    DecodeOptions.OptionalTasks = CompletedTasks;
```

```
    %---Optionally save---
    if( SaveRequired && FileOptions.SaveResult )
        if( isfloat(LF) )
            LF = uint16( LF .* double(intmax('uint16')) );
        end
        ThumbFname = sprintf(FileOptions.ThumbFnamePattern, LFFnameBase);
        fprintf('Saving to:¥n¥t%s,¥n¥t%s...¥n', SaveFname, ThumbFname);
        TimeStamp = datestr(now,'ddmmmyyyy_HHMMSS');
        GeneratedByInfo = struct('mfilename', mfilename, 'time', TimeStamp, 'VersionStr', LFToolboxVersion);

        save('-v7.3', SaveFname, 'GeneratedByInfo', 'LF', 'LFMetadata', 'WhiteImageMetadata', 'LensletGridModel', 'DecodeOptions', 'RectOptions');
        imwrite(Thumb, ThumbFname);
    end
end
end

%-------------------------------------------------------------------------------------------------
function  [SDecoded, FileExists, CompletedTasks, TasksRemaining, SaveFname] = ...
    CheckIfExists( LFFnameBase, DecodeOptions, SaveFnamePattern, ForceRedo )

SDecoded = [];
FileExists = false;
SaveFname = sprintf(SaveFnamePattern, LFFnameBase);

if( ~ForceRedo && exist(SaveFname, 'file') )
    %---Task previously completed, check if there's more to do---
    FileExists = true;
    fprintf( '    %s already exists¥n', SaveFname );

    PrevDecodeOptions = load( SaveFname, 'DecodeOptions' );
    PrevOptionalTasks = PrevDecodeOptions.DecodeOptions.OptionalTasks;
    CompletedTasks = PrevOptionalTasks;
    TasksRemaining = find(~ismember(DecodeOptions.OptionalTasks, PrevOptionalTasks));
    if( ~isempty(TasksRemaining) )
        %---Additional tasks remain---
        TasksRemaining = {DecodeOptions.OptionalTasks{TasksRemaining}};  % by name
        fprintf('    Additional tasks remain, loading existing file...¥n');

        SDecoded = load( SaveFname );
        AllTasks = [SDecoded.DecodeOptions.OptionalTasks, TasksRemaining];
        SDecoded.DecodeOptions.OptionalTasks = AllTasks;

        %---Convert to float as this is what subsequent operations require---
        OrigClass = class(SDecoded.LF);
        SDecoded.LF = cast( SDecoded.LF, SDecoded.DecodeOptions.Precision ) ./ ...
            cast( intmax(OrigClass), SDecoded.DecodeOptions.Precision );
        fprintf('Done¥n');
    else
        %---No further tasks... move on---
        fprintf( '    No further tasks requested¥n');
        TasksRemaining = {};
    end
else
    %---File doesn't exist, all tasks remain---
    TasksRemaining =  DecodeOptions.OptionalTasks;
    CompletedTasks = {};
end
end

%-------------------------------------------------------------------------------------------------
function Thumb = DispThumb( LF, CurFname, CompletedTasks)
Thumb = squeeze(LF(floor(end/2),floor(end/2),:,:,:)); % including weight channel for hist equalize
Thumb = uint8(LFHistEqualize(Thumb).*double(intmax('uint8')));
Thumb = Thumb(:,:,1:3); % strip off weight channel
LFDispSetup(Thumb);
Title = CurFname;

for( i=1:length(CompletedTasks) )
    Title = [Title, ', ', CompletedTasks{i}];
end

title(Title, 'Interpreter', 'none');
drawnow
end

%-------------------------------------------------------------------------------------------------
function LF = ColourCorrect( LF, LFMetadata, DecodeOptions )
fprintf('Applying colour correction... ');

%---Weight channel is not used by colour correction, so strip it out---
LFWeight = LF(:,:,:,:,4);
LF = LF(:,:,:,:,1:3);
```

```
%---Apply the color conversion and saturate---
LF = LFColourCorrect( LF, DecodeOptions.ColourMatrix, DecodeOptions.ColourBalance, DecodeOptions.Gamma );

%---Put the weight channel back---
LF(:,:,:,:,4) = LFWeight;

end

%--------------------------------------------------------------------------------------------
function [LF, RectOptions, Success] = Rectify( LF, LFMetadata, DecodeOptions, RectOptions, LensletGridModel )
Success = false;
fprintf('Applying rectification... ');
%---Load cal info---
fprintf('Selecting calibration...¥n');

[CalInfo, RectOptions] = LFFindCalInfo( LFMetadata, RectOptions );
if( isempty( CalInfo ) )
    warning('No suitable calibration found, skipping');
    return;
end

%---Compare structs
a = CalInfo.LensletGridModel;
b = LensletGridModel;
a.Orientation = strcmp(a.Orientation, 'horz');
b.Orientation = strcmp(b.Orientation, 'horz');
FractionalDiff = abs( (struct2array(a) - struct2array(b)) ./ struct2array(a) );
if( ~all( FractionalDiff < RectOptions.MaxGridModelDiff ) )
    warning(['Lenslet grid models differ -- ideally the same grid model and white image are ' ...
        ' used to decode during calibration and rectification']);
end

%---Perform rectification---
[LF, RectOptions] = LFCalRectifyLF( LF, CalInfo, RectOptions );
Success = true;
end
```