An Introduction to the Piecewise Parabolic Method (PPM) Library
II. I/O and Data Analysis

David H. Porter,
Sarah E. Anderson,
Paul R. Woodward,
and B Edgar,

Laboratory of Computational Science or Engineering,
Department of Astronomy, School of Physics and Astronomy
116 Church St. SE.
University of Minnesota
Minneapolis, Minnesota 55455

6. LCSE Movie Maker

a. Introduction

The LCSE Movie Maker software suite is a collection of programs which ease the onerous task of creating volume rendered movies from a data set. These programs operate on the ''Brick of Byte'' formats produced by the LCSE analysis tool **A3D** discussed in previous sections of the guide. This data format consists of a 3-dimensional array where each array element corresponds to a computational zone or *voxel* computed on a regular Cartesian grid. Usually each array element is one byte, although four bytes per voxel is also supported. **(Is this so?)** The array is stored with the x-direction index varying fasted and the z-direction index the slowest varying. No header information is stored in the files. Users with curvilinear, irregular, or unstructed grids may need to preprocess their data and interpolate to create an equivalent data set for a uniform grid. This software is still under development, on new functionality will appear soon. It is the intent that this will be an evolutionary process rather and a revolutionary process so that experience with the current version is compatible with future releases.

The user may specify complex paths not only in space but but also in time if given a time sequence of individual data files. This path may include the entire data volume, or the user may zoom in to examine closely small, curious regions. The user controls the resolution of the final images (in screen pixels) as well as the number of images (movie frames) along the chosen path. This encourages coarse previews and data overviews before undertaking a time consuming high resolution movie.

The individual images are created using the the technique of *alpha blending.* Each data voxel has, in addition to the usual three values defining colour (red, green, blue) a fourth value corresponding to the degree of transparency of the values. When projected onto the display screen, the three dimensional data voxels become 2-dimensional polygons. When several polygons with differing alphas are overwritten (in the alpha channel or alpha buffer) a degree of transparency is achieved. This is often used in computer graphics for depth-cueing or to simulate an atmospheric effect such as fog. It is also ideally suited for creating images of substances such as fluids which are often expensive to render using classic ray-tracing techniques due to their lack of opac surfaces.

programs depend upon the alpha blending capabilities of the SGI workstations. For each voxel, an alpha blended polygon is created in the alpha buffer (a special section of memory). The polygons are drawn from the back of the brick of bytes to its front as seen from the observers viewpoint. Rather than using an external light source (as would be expected in a standard ray-tracing technique) the user assigns colours and alphas to specific data values. This is done using a user-specified look up table. This simulates what the user would see if his data consisted of a brick of voxels of glowing gases which have differing opacities.

The current version may the obtained over the Web at]fBwww.lcse.umn.edu/PPMlib/MovieMaker. There will be found three subdiretories:

Bin -- This contains a gzipped tar file *lcse_moviemaker.tar.gz* which contains the executables and scripts necessary. The executables have been compiled for SGI O2K systems.

Data -- This contains sample data files used to illustrate operation of the programs. Also located here is the source code to the program which generated these data files.

Movie -- Here are the resulting movies created by the examples discussed below.

It is recommended, but not required, that a similar directory structure be retained to reduce confusion. The user will quickly discover to his horror that during operation, the programs create innumerable scratch files and directories (in addition to the movies themselves) in whatever directory directed by the the user environmental variable ‘‘BinDir’’. Likewise, the programs search for input data only in the directory pointed to by the user environmental variable ‘‘BrickDir’’. The user environmental variable‘BinDir’’ provides the search path of locating necessary programs and scripts.

It is necessary to have TCL (Tool Command Language, a popular open source scripting language) and TK (a GUI toolkit) installed on your system. These may be found using http://www.scriptics.com/software/8.0.html. Specifically, the required tar files are

ftp://ftp.scriptics.com/pub/tcl/tcl8_0/tcl8.0.3.tar.gz

and

ftp://ftp.scriptics.com/pub/tcl/tcl8_0/tk8.0.3.tar.gz.

A responsible user would then which to consult his system administrator for further instructions.

In addition to the two user environment variables mentioned above, A few others are also necessary. These are

BrickDim: This specifies the 3 coordinate dimensions of the data bricks in the format NXxNYxNZ where NX, NY, and NZ are the integer dimensions in the x-, y-, and z-directions, respectively.

BobHead: This is an alpha-numeric ASCII string used to construct data file names.

BobTail: This is an alpha-numeric ASCII string used to construct data file names.

RenderTail: This is an alpha-numeric ASCII string used to construct data file names.

It is assumed that the format of the file names is `hhhhhh####tttttt` where `hhhhhh` is the alpha-numeric character string assigned to the `BobHead`, #### is a four digit number ranging from 0000 to 9999, and `tttttt` is the alpha-numeric character string assigned to `BobTail` or `RenderTail`. These two different file name suffixes provide flexibility when dealing with very large data bricks. In such circumstances it is advantageous to construct a smaller set by blending together zones or subsampling the initial data. The resulting data files should be given the file suffix assigned to `BobTail`. During key frame selection, these reduced files will be used create the snapshots of the selected keyframes. This process will proceed much faster on the smaller data bricks. During the actual movie rendering however, the files with the same suffix as that assigned to `RenderTail` will be used to produce a high resolution movie. Both of these suffixes may be the same and generally for small data sets are. They may also be blank, but they *must* be assigned something. When appropriate, numerical sequence in the file names will be used to specify the temporal ordering of files when a movie combines both spatial and temporal paths. Finally, The Movie Maker suite currently handles up to four RE (**Whatis “RE”? Reality Engines what are they usually called in the literature?).** The list of managed IR (**WHAT IS “IR”?**) pipes must be (**WHAT?? - defined assigned created**). This is also done using user environmental variables ‘‘PIPE0’’, ‘‘PIPE1’’, ‘‘PIPE2’’, and ‘‘PIPE3’’. To specify REs (Reality Engines), the user would enter the appropriate raster manager (e.g. unix:0.0) on his system, leaving unmanaged or non-existent pipes empty.

The above steps are perhaps best performed as a Tcl script. An example, named*SETUP*, of such a script file is provided in the Movie Maker tar file:

```
#
#  File "SETUP" is an example TCL script file which performs the
#  various setup chores necessary before one can run the LCSE Movie
#  Maker Suite.  The user is expected to make appropriate changes to
#  the values of the user environmental variables below.
#
#  $BrickDir is the directory with data files:
 setenv BrickDir /home/bke/PPMLIB/MIRROR/PUBLIC/Movie_maker/Example
#
#  BrickDim is the dimensions of data files: NXxNYxNZ
#  here, a cube of 64**3 zones
 setenv BrickDim 64x64x64
#
#  $BinDir is the directory with scripts & executables
 setenv BinDir /export/prune/wrinkled/moviebin
#
#  $BobHead is the first part of the names  of the desired data files.
#  $BobTail is the last part of the names  of the desired data files.
#  $RenderTail is the last part of the names  of the desired data files.
#    Normally, $BobTail and $RenderTail are identical.  However, with
#    very large data sets (e.g. bricks 1024x1024x1024 or larger ) selecting
#    key frames goes much faster is a smaller brick size is used.  This
#    can be created either by subsampling or blending.  The actual rendering
#    is performed using the high resolution data set. $BobTail and $RenderTail
#    allow the user to differentiate between these two sets.
#    The data files $BrickDir/$BobHead####$BobTail  will be used for selecting
#    the key frames, while the data files $BrickDir/$BobHead####$RenderTail
#    will be used for for rendering.
#    In the above, #### is a four character number in the range 0000 to 9999,
#  e.g. /export/mercury/dhp/data/p14aG0013
 setenv BobHead p14a
 setenv BobTail
 setenv RenderTail
#
#  The following is a list of managed IR pipes:
 setenv PIPE0 unix:0.0
 setenv PIPE1
 setenv PIPE2
 setenv PIPE3
#
#  NOTE:  You need to modify the file SETUP to reflect your data and
#  the configuration of your system (i.e., the number of IR pipes and
#  names of corresponding raster managers -- which you can find by using
#  /usr/gfx/gfxinfo).
#
#  The scripts are currently set up to handle as many as 4 REs -- they can
#  easily be modified to handle any number.  To specify REs (Reality Engines)
#  for your system, enter their raster managers (like unix:0.0).  Leave
#  unmanaged or non-existent   pipes empty (e.g., "setenv PIPE2" above).
#
#  "rundokey" is the name of script with runs the LCSE Movie Maker, this alias
#  is merely a convenience for the user.
 alias  rundokey $BinDir/dokey
#
#  This file must be evaluated as a Tcl script via _source_.  That is
#
# source SETUP
#
#  You need to source SETUP rather than run it as a shell script so that the
#  environment variables (above) are set properly in the shell from which you
#  intend to run "rundokey".
#
```

Note, that as a convenience the command “rundokey” has been aliased to the appropriate script which executes the Movie Maker. Also included in the tar file is the directory “Example-data” which contains the input data (bricks of bytes) used in the examples discussed below. These data are created by the program `moivemakerexample` which is also included. This program is also provided with the data bricks.

The data set consists of a series of 64x64x64 cubic bricks of bytes. The initial brick, `p14a0000`, consists of a 2x3x5 array of elliptical blobs (figure 3). The finial brick of the series `p14a0100`, consists of the same array of blobs plus a jack which elliptical cross-sections (Figure 4) A series (mimicing a time series) is created by increasing the amplitude of the central jack from zero (`p14a0000`) to 255 (`p14a0100`).

The capabilities of the LCSE Movie Maker program suite will be demonstrated by a number with simple examples, for which it will be assumed that the user has obtained the above sample data and set appropriately the necessary environmental variables. For the brave, hardy few of our readers remaining, turn us now to our first example.

This first example is intended to familiarize the user with the MovieMaker environment.

Entering the command `rundokey` should display three windows as illustrated in figure 5. The first (figure 5a) is the keyfinder. In the viewport portion will appear a three-dimensional rendered image of the data. Initially, only a small 32x32x32 cube located in the front lower left corner of the data brick will be displayed. To the right of the viewport will be a listing of all files whose names match #Bob-Head####Bottail as discussed above. A set of button appear in the top right portion. The only button needed is the one containing the small octagon. This button is called the Key Frame Selector. As the user manipulates his data, key frames are selected by clicking on this button. A key frame is an interesting vantage point which the user may wish to use to define a path through the data. An opportunity to select, reject and reorder key frames will occur later, so order is unimportant at this point. The other buttons exist to confuse the uninitiated. You may ignore them as well as the man behind the curtain. The second window (Viewing Box Selector, figure 5b) allows the user to adjust the dimensions and locations of the viewing box withing the data cube. (It is recommended that when working with large data volumes, the stride control in the Viewing Box Selector be set greater than one. This will accelerate the rendering of the image in the Data viewport, while having no effect on the quality of the rendered movie. Only the size and location of the viewing Box matter.)

The coordinate system used is illustrated in figure 6. There are two coordinate systems, the three dimensional object coordinate system, represented by the xyz unit vector triad, and the two dimensional system on the projected Viewing Plane, represented by the X,Y vector pair. The viewing plane is divided into N_x *x* N_y *pixels* or picture elements. The viewpoint of the observer is represented by the eyepoint E, the look point is point L. this defines the gaze vector G,

$$\mathbf{G} = \mathbf{L} - \mathbf{E}.$$

The half vertical field of view (fov) is V and whose default value is 17^o. The half horizontal fov, H, is set to retain a square aspect ratio. That is,

$$H = V * \frac{N_x}{N_y}\ .$$

The viewing plane is orthogonal to the gaze vector. The direction of the viewing plane y-axis is parallel to the direction of the up vector U. This defines a viewing frustum with the viewing plane at its base and the eyepoint at the apex. the viewing plane at its base and the eyepoint at the apex.

By default, U is in the direction of the y-axis of the object coordinate system. The user along which of the three coordinate axes to point the upvector by selecting “Flip X”, “Flip Y”, or, “Flip Z” from the “VIEW” pulldown menu on the Keyfinder. Figure 7a illustrates the result of selecting the x-coordinate direction for the up vector.

The length of the gaze vector is altered by moving the cursor into the Data viewport portion of the Keyfinder and smartly depressing the right mouse button. Dragging the cursor down the screen lengthens the gaze vector (causing the image to appear smaller) while dragging the cursor up shortens the gaze vector (causing the image to appear larger), as illustrated in figure 7b.

The gaze vector may be rotated about the eyepoint by depressing the middle mouse button while the cursor is the in the data viewport. As illustrated by figure 7c, the apparent motion of the object on the screen is opposite that of the gaze vector. If the gaze vector is rotated so that the lookpoint moves from above the object to below, the object appears to move upwards on the screen. The gaze vector may also be rotated about the lookpoint by pressing on the left mouse button while the cursor is in the data viewport. Figure 7d illustrates how an anticlockwise rotation of the gaze vector about the lookpoint produces an apparent clockwise rotation of the object.

The third window to appear (figure 5c) is the Movie Maker control panel. Thumbnail representations of the selected key points will appear in the middle section when the "more keys" button is clicked. Old key frames may be cleared by the "clear keys" button. This will reinitialize the entire key frame selection process. The path is the virtual observer through and about the data is determined by selecting desired key frames which then appear in the bottom section. Between each selected key frame, the user may specify the number of viewpoints to insert and may also specify some number of rotations of the data. A smooth path is then interpolated through these viewpoint, looping back to the beginning to complete a closed loop. This will be described in detail later in the examples.

Movie Examples