Radiance Discussion

Greg Ward, Taoning Wang, All

Topics for final session

1. *Radiance* refactoring effort & progress
2. IBPSA willingness to administer funding
3. Open discussion

* Improved upon *RtraceSimulManager* class and **rxtrace** demonstration tool
* Created new *RpictSimulManager* class and **rxpict** demonstration tool
* Looking into more ambitious reimagining of **rcontrib**-related class, more on that next year

1. *Radiance* refactoring effort & progress

RtraceSimulManager class methods (1)

```
/// Ray reporting callback method -- returns # successfully reported, -1 to abort
typedef int     RayReportCall(RAY *r, void *cd);

                        /// Load octree and prepare renderer
bool                    LoadOctree(const char *octn);
                        /// Prepare header from previous input (or clear)
bool                    NewHeader(const char *inspec=NULL);
                        /// Add a line to header (adds newline if none)
bool                    AddHeader(const char *str);
                        /// Append program line to header
bool                    AddHeader(int ac, char *av[]);
                        /// Get header lines or empty string
const char *            GetHeader() const;
                        /// How many cores are available?
static int              GetNCores();
                        /// Set number of computation threads (0 => #cores)
int                     SetThreadCount(int nt = 0);
                        /// Check thread count (1 means no multi-threading)
int                     NThreads() const;
                        /// How many threads are currently unoccupied?
int                     ThreadsAvailable() const;
                        /// Are we ready?
bool                    Ready() const;
```

RtraceSimulManager class methods (2)

```
                        /// Process a ray (in subthread), optional result
bool                    ProcessRay(RAY *r);
                        /// Wait for next result (or fail)
bool                    WaitResult(RAY *r);
                        /// Add ray bundle to queue w/ optional 1st ray ID
int                     EnqueueBundle(const FVECT orig_direc[], int n,
                                      RNUMBER rID0 = 0);
                        /// Enqueue a single ray w/ optional ray ID
bool                    EnqueueRay(const FVECT org, const FVECT dir,
                                   RNUMBER rID = 0);
                        /// Set/change cooked ray callback
void                    SetCookedCall(RayReportCall *cb, void *cd = NULL);
                        /// Set/change trace callback
void                    SetTraceCall(RayReportCall *cb, void *cd = NULL);
                        /// Finish pending rays and complete callbacks (return
                         #sent)
int                     FlushQueue();
                        /// Close octree, free data, return status
int                     Cleanup(bool everything = false);
```

rxtrace demonstration tool

* Nearly identical syntax and semantics as **rtrace** minus persist (**-P** and **-PP**) options
* Queues rays and produces results using *RtraceSimulManager* class
* Supports same 6.0 set of hyperspectral rendering and output options

RpictSimulManager class methods (1)

```
                        /// Load octree and prepare renderer
bool                    LoadOctree(const char *octn);
                        /// Prepare header from previous input (or clear)
bool                    NewHeader(const char *inspec=NULL);
                        /// Add a string to header (adds newline if none)
bool                    AddHeader(const char *str);
                        /// Append program line to header
bool                    AddHeader(int ac, char *av[]);
                        /// Get header lines if any
const char *            GetHeader() const;
                        /// Set number of computation threads (0 => #cores)
int                     SetThreadCount(int nt = 0);
                        /// Check thread count (1 means no multi-threading)
int                     NThreads();
                        /// How many threads are currently unoccupied?
int                     ThreadsAvailable() const;
                        /// Are we ready?
bool                    Ready() const;
                        /// Close octree, free data, return status
int                     Cleanup(bool everything = false);
```

Methods shared from RtraceSimulManager parent class

RpictSimulManager class methods (2)

```
ProgReportCB *          prCB;                   // progress report call-back
RGBPRIMP                prims;                  // output primaries (NULL if
 spectral)
int                     frameNo;                // frame number (0 if not sequence)

                        /// Assign reference depth string (e.g., "2.5/meter")
bool                    SetReferenceDepth(const char *dstr);
bool                    SetReferenceDepth(double dref, const char *unit=NULL);
                        /// Return reference depth
double                  GetReferenceDepth(char *du=NULL) const {
                                if (du) strcpy(du, dunit);
                                return pacc.refDepth;
                        }
                        /// Set up rendering frame (call after octree loaded)
                        /// Overall dimensions may be adjusted for view,
                        /// optional pixel aspect ratio and tile grid
                        /// Increments frameNo if >0
bool                    NewFrame(const VIEW &v, int xydim[2], double *ap=NULL,
                                const int *tgrid=NULL);
                        /// Get current view if set
const VIEW *            GetView() const;
                        /// Writeable previous view (for motion blur)
VIEW &                  PreView();
```

RpictSimulManager class methods (3)

```
                        /// Get current picture width
int                     GetWidth() const;
                        /// Get current picture height
int                     GetHeight() const;
                        /// Tile width
int                     TWidth() const;
                        /// Tile height
int                     THeight() const;
                        /// Render the specified tile in frame
                        /// Tile pixels are contiguous unless ystride != 0
                        /// Tiles numbered from lower-left at (0,0)
                        /// Pixel type influenced by this->prims assignment
bool                    RenderTile(COLORV *rp, int ystride=0, float *zp=NULL,
                                const int *tile=NULL);
                        /// Same but store as common-exponent COLR or SCOLR
bool                    RenderTile(COLRV *bp, int ystride=0, float *zp=NULL,
                                const int *tile=NULL);
                        /// Same but also use 16-bit encoded depth buffer
bool                    RenderTile(COLRV *bp, int ystride, short *dp,
                                const int *tile=NULL);
                        /// Back to float color with 16-bit depth
bool                    RenderTile(COLORV *rp, int ystride, short *dp,
                                const int *tile=NULL);
```

RpictSimulManager class methods (4)

```
                        /// Render and write a frame to the named file
                        /// Include any header lines set prior to call
                        /// Picture file must not exist
                        /// Write pixels to stdout if !pfname
                        /// Write depth to a command if dfname[0]=='!'
RenderDataType          RenderFrame(const char *pfname,
                                    RenderDataType dt=RDTrgbe,
                                    const char *dfname=NULL);
                        /// Resume partially finished rendering
                        /// Picture file must exist with valid header
RenderDataType          ResumeFrame(const char *pfname,
                                    const char *dfname=NULL);
                        /// Prepare new picture (and depth) output
                        /// Called by RenderFrame()
RenderDataType          NewOutput(FILE *pdfp[2], const char *pfname,
                                    RenderDataType dt=RDTrgbe,
                                    const char *dfname=NULL);
                        /// Reopen existing picture (and depth) file
                        /// Called by ResumeFrame()
                        /// File pointers @ end of header (before res.)
RenderDataType          ReopenOutput(FILE *pdfp[2], const char *pfname,
                                    const char *dfname=NULL);
```

RpictSimulManager class types

```
/// Data type flags for pixel access and output
enum RenderDataType {
        RDTnone=0,
        RDTscolor=0x1, RDTrgb=0x2, RDTxyz=0x3, RDTscolr=0x4, RDTrgbe=0x5,
         RDTxyze=0x6,
        RDTcolorM=0x7,
        RDTdfloat=0x8, RDTdshort=0x10,
        RDTdepthM=0x18
};
```

rxpict demonstration tool

* Similar syntax and semantics to rpict, again without persist (**-P** and **-PP**) options
* Employs *RpictSimulManager* class
* Adds native multi-processing with **-n** option
* Adds spectral output with **-co+** option, either as HSR picture or float matrix

* Discussion with IBPSA leadership last Thursday
* Indicated they could manage small(ish) donations from companies to fund *Radiance* development efforts
* May offer a way to pay individuals for important updates/contributions (*cough*photonmap*)

2. IBPSA willingness to administer funding

3. Open discussion