New *Radiance* 3.5 Features

Greg Ward
Anyhere Software

Radiance 3.5

- Released March 21, 2003
- Source-level changes:
  - Adds ANSI-C function prototypes to headers
  - Provides rendering library functionality
- New system features:
  - New **ranimove** program
  - New *mesh* primitive
  - Includes holodeck system introduced in 3.4 (**rholo** and friends)

Source-level Changes

- ANSI-C function prototypes
  - Guards against parameter passing errors in new code additions
  - Example:

```
void    ray_trace(RAY *r);
```

- Radiance as C-callable rendering library
  - Avoids need for separate **rtrace** processes
  - Provides enhanced control over calculation and possible incorporation as DLL under Windows

Serial Library Interface

void	ray_init(char *otnm);	Load octree and initialize
void	ray_trace(RAY *r);	Trace a ray
void	ray_done(int freall);	Unload octree and free
void	ray_save(RAYPARAMS *rp);	Record parameter settings
void	ray_restore(RAYPARAMS *rp);	Restore parameter settings
void	ray_defaults(RAYPARAMS *rp);	Restore defaults

Parallel Library Interface

void	ray_pinit(char *otnm, int nproc);	Initialize parallel calculation
void	ray_psend(RAY *r);	Add a ray to rendering queue
int	ray_pqueue(RAY *r);	Add a ray and check queue
int	ray_presult(RAY *r, int poll);	Get a ray result from queue
void	ray_pdone(int freall);	End parallel calculation
void	ray_popen(int nadd);	Start parallel process(es)
void	ray_pclose(int nsub);	Close parallel process(es)

Rendering Library Tradeoffs

- Current interface provides abstract layer to control rendering parameters and operation
- However, only one rendering context supported at a time
- Calling program is not insulated from *Radiance* namespace
- A couple of global variables still set by caller (progname, trace)

Longer Term Goals

- Provide rendering context (or C++ class) that bundles scenes with their parameters
- Create a more logical separation between general and ray-tracing specific parts of *Radiance* and its associated tools
- The ultimate library would incorporate all of *Radiance*'s functionality -- is such a thing even feasible?

Features Added Since *RwR*

I. Holodeck Rendering (ver. 3.4)
   - A continually progressive light field rendering method permitting users to move freely

II. The *mesh* Primitive (ver. 3.5)
   - Handles complex, smooth geometry with local (u,v) texture coordinates

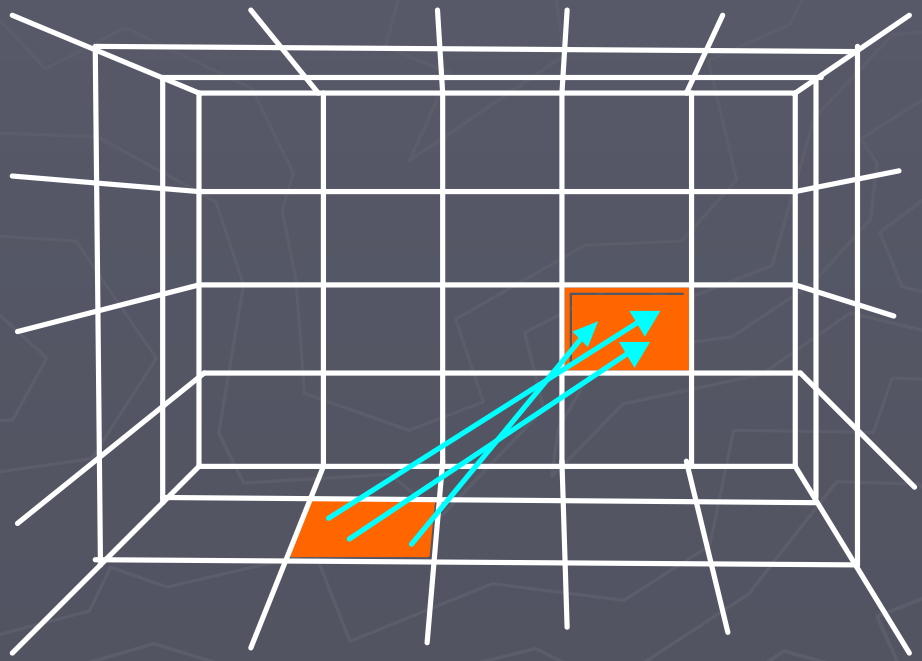
III. The **ranimove** Program (ver. 3.5)
   - Fast animation rendering with full motion blur

I. Holodeck Rendering

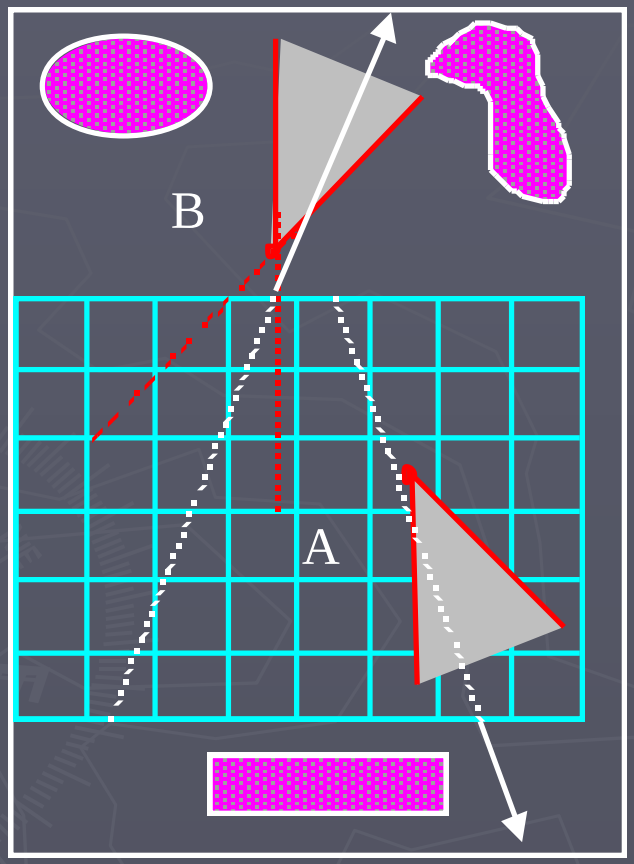
- Ray tracing with interactive display
- Accurate global and local illumination
- Interactive walk-throughs of non-diffuse environments
  - display representation for motion feedback
- No wasted computation
  - i.e., don’t throw away what we’ll want again

The Holodeck Ray Cache

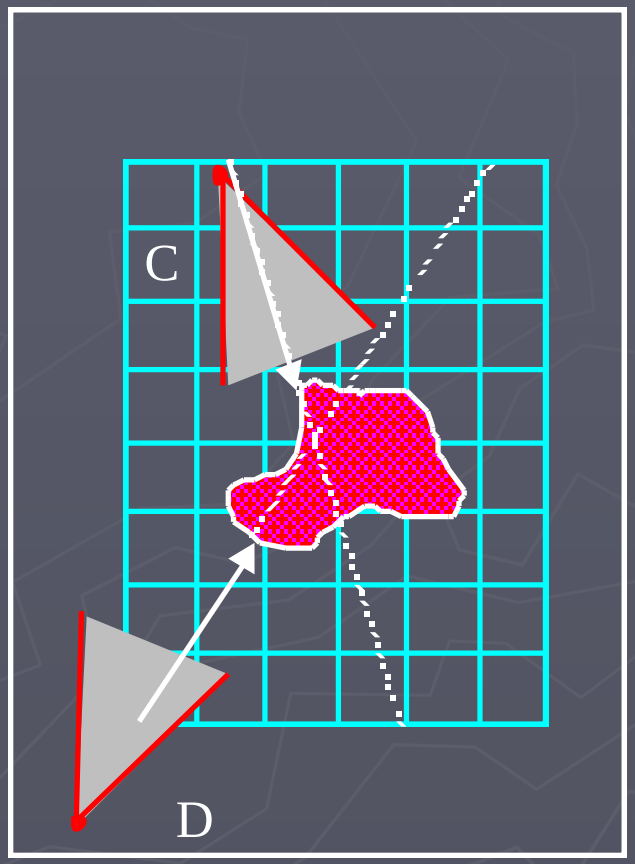
- Rays bundled in *beams* going in one *cell* and out another on a *section wall*



View-Beam Correspondence

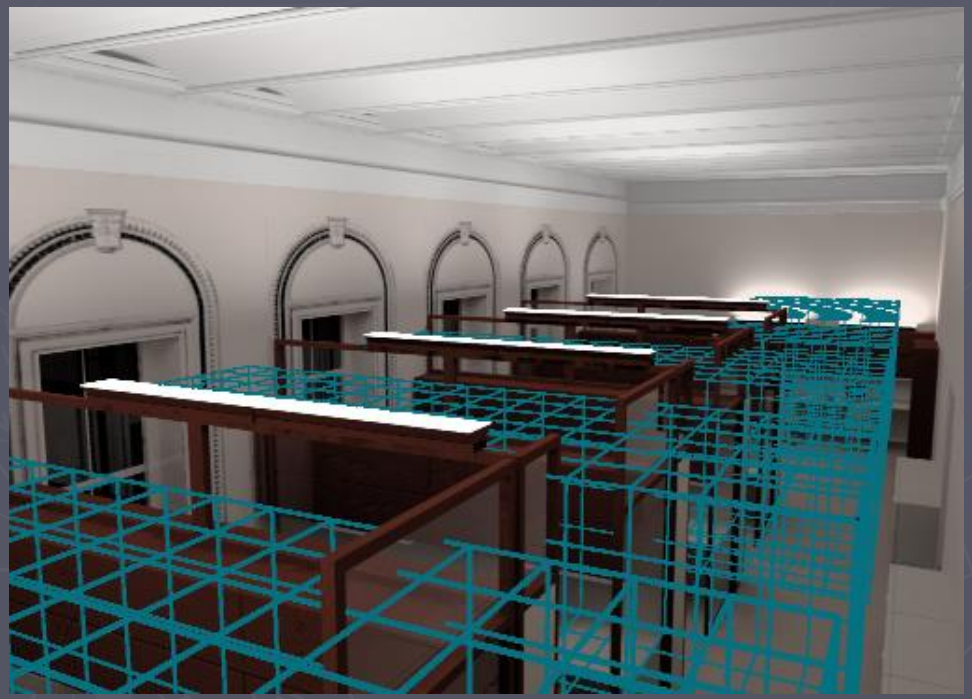


“Internal” Section



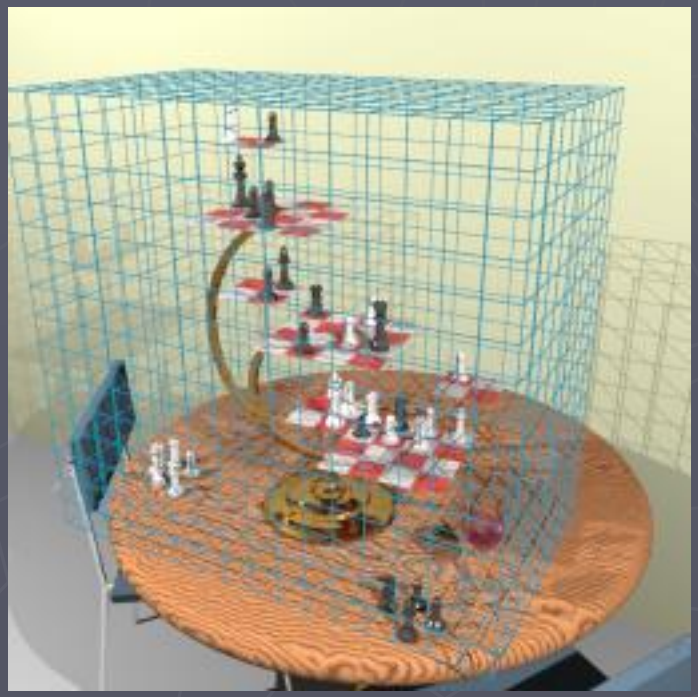
“External” Section

Example Holodecks



Internal Sections

(view from inside)



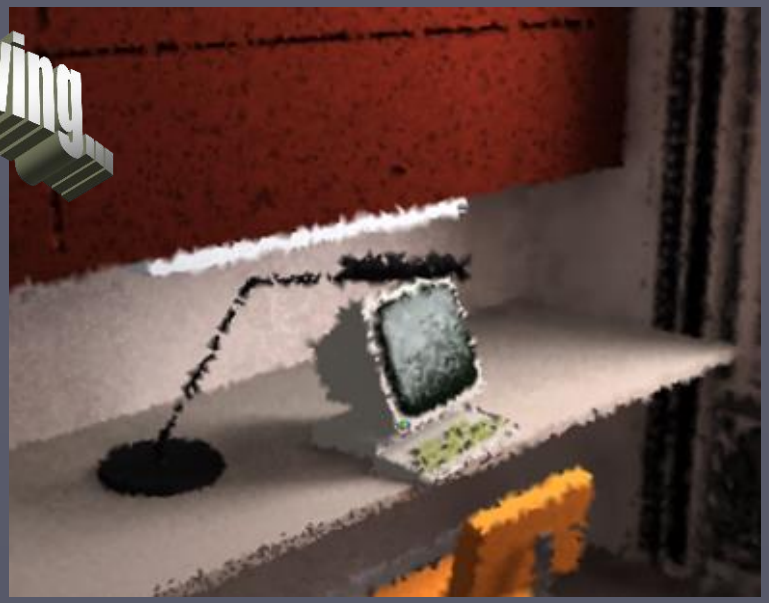
External Section

(view from outside)

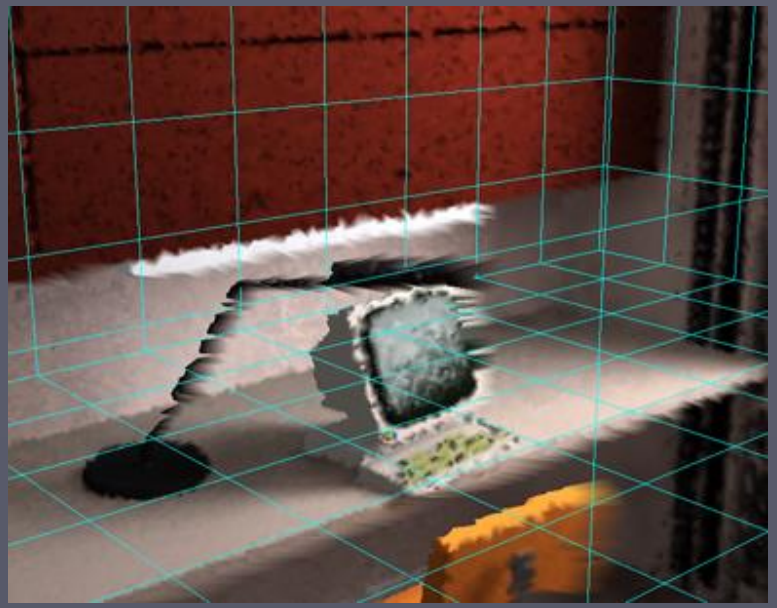
Progressive Rendering

SP Octane after 10 seconds...

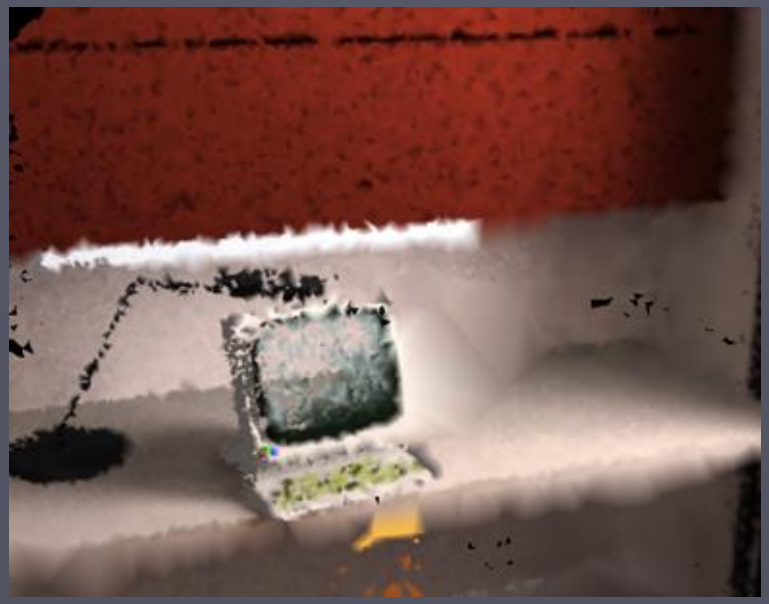
after 1 minute



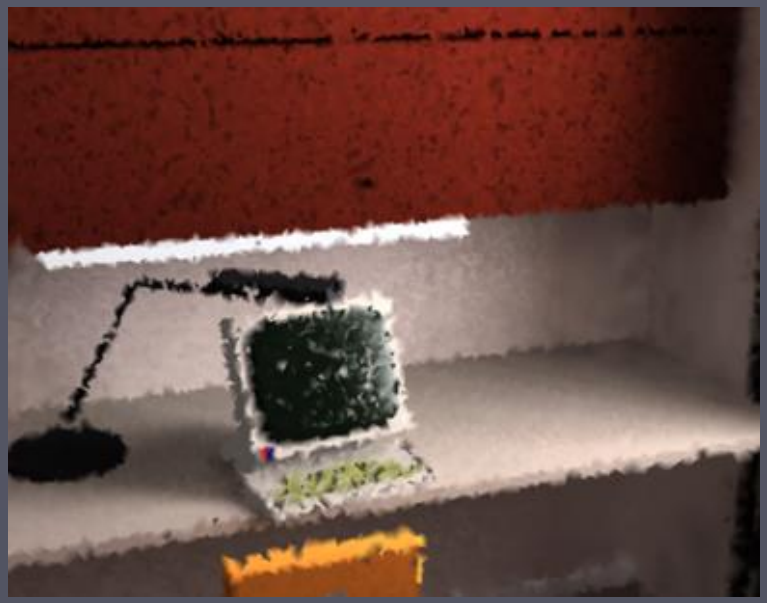
starting view



begin move...



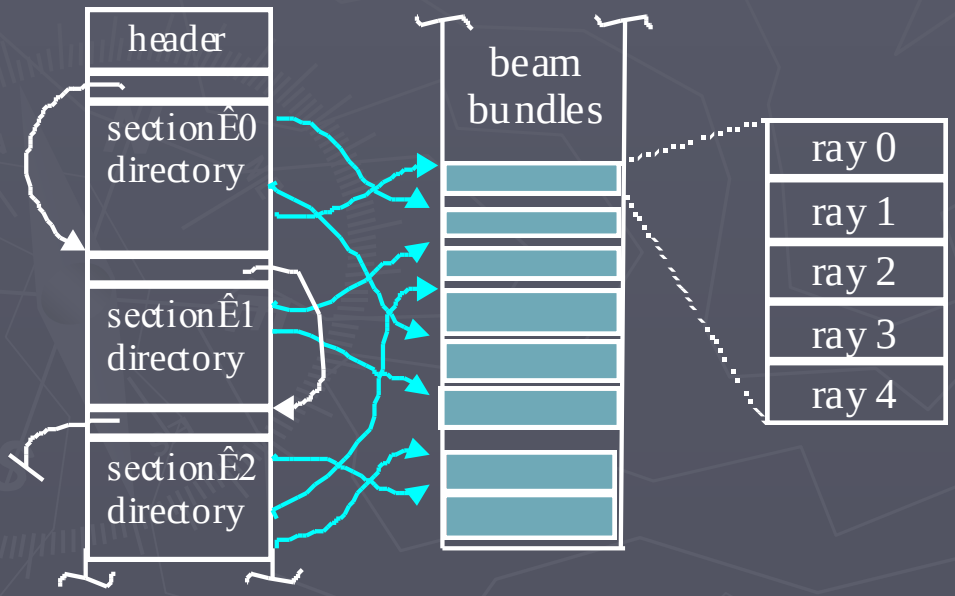
...end move



new samples come in

Holodeck File Structure

- Holodeck may have multiple sections
  - each section has its own directory
  - each beam sample (ray) takes 10 bytes

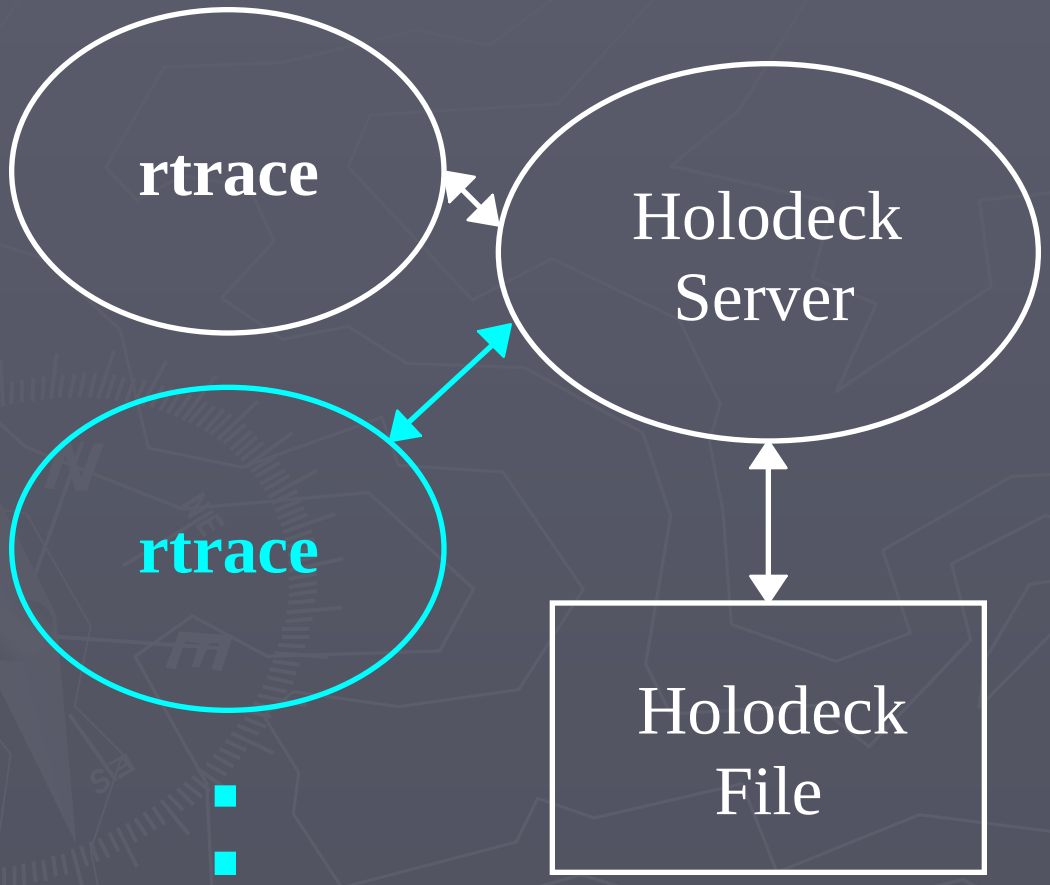


Encoded value	Size
floating point color	4 bytes
position in starting cell	2 bytes
position in ending cell	2 bytes
ray distance	2 bytes

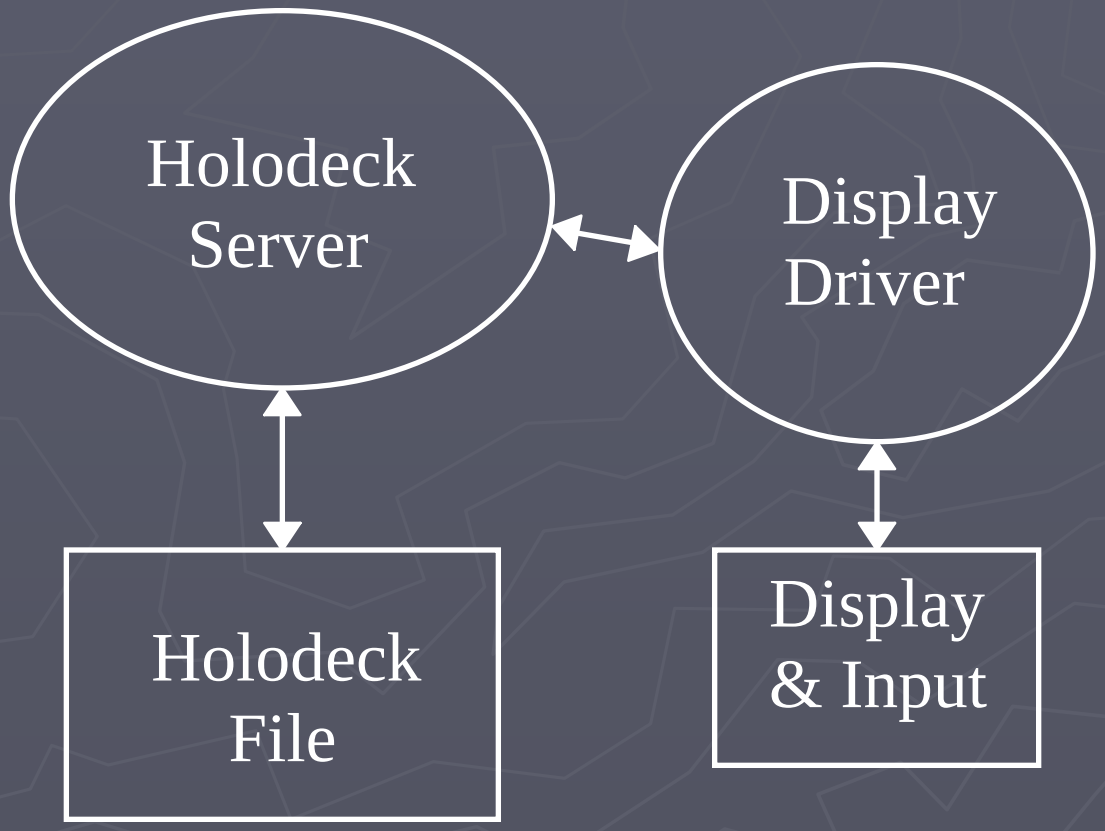
Holodeck Operation

- Holodeck server manages holodeck file
- Display driver manages what the user sees and does
- Sample generator(s) compute new ray samples to use in holodeck

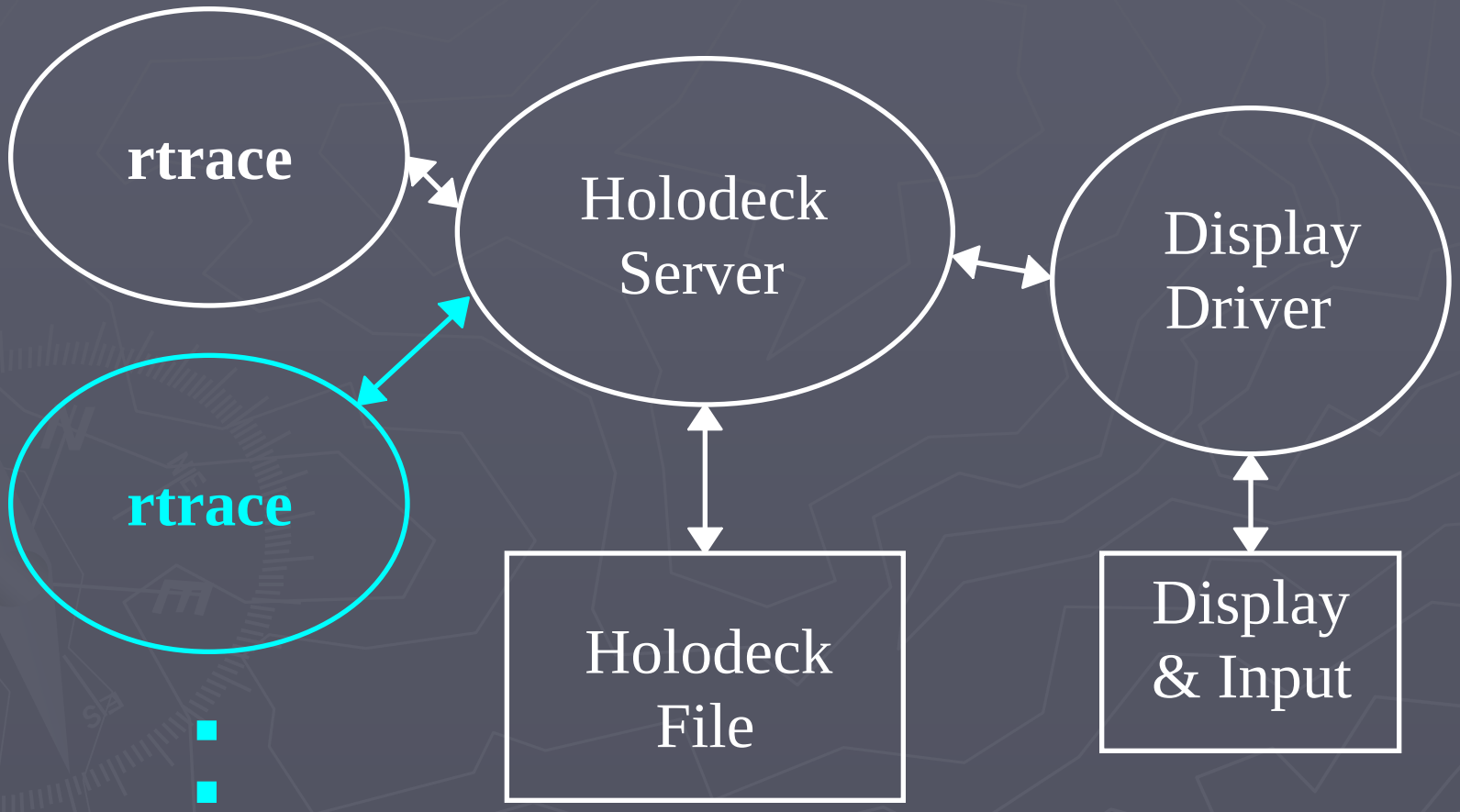
Batch Rendering Mode



Display Only Mode



Interactive Rendering



Subproblems to Solve

- Holodeck cache management
  - managing beam requests
  - LRU beam replacement scheme
- Parallel rendering
  - process synchronization and data sharing
- User interaction and display
  - determining which beams to request
  - display representation and tone mapping

Holodeck Cache Management

- Sort and maintain beam request list
  - Nrequested/(Ncomputed+1) priority
- Load data from holodeck into memory
- When memory cache limit is reached, use LRU scheme to free space
- As beam sizes grow, maintain file fragment list to optimize disk usage
- Recover from system errors

Parallel Rendering

- Coarse-grained parallelization
  - multiple ray tracing processes sharing memory and data as much as possible
- Beam packets assigned to maintain maximum queue size on each process
- Packets returned to server, which caches them and passes them on to display driver

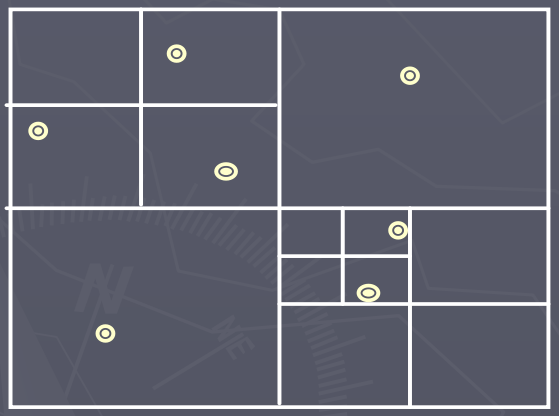
User Interaction and Display

- Get user input
  - view-motion and control commands
- Determine beams corresponding to view
  - sparse, jittered view sampling
  - should be stable for small motions
- Display beam sample data
  - need 2.5-D intermediate representation
  - fill in gaps & resolve multidepth samples

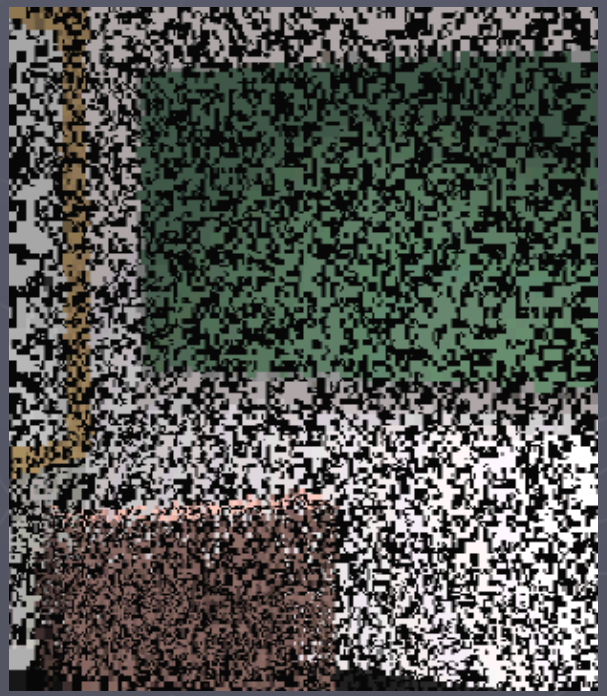
Display Representation

- Quadtree representation w/ X11
  - simple, fast, ugly
- Vornoi cell representation w/ OpenGL
  - simple, almost as fast, not as ugly
- Spherical Delaunay mesh w/ OpenGL
  - not simple, pretty fast, less ugly

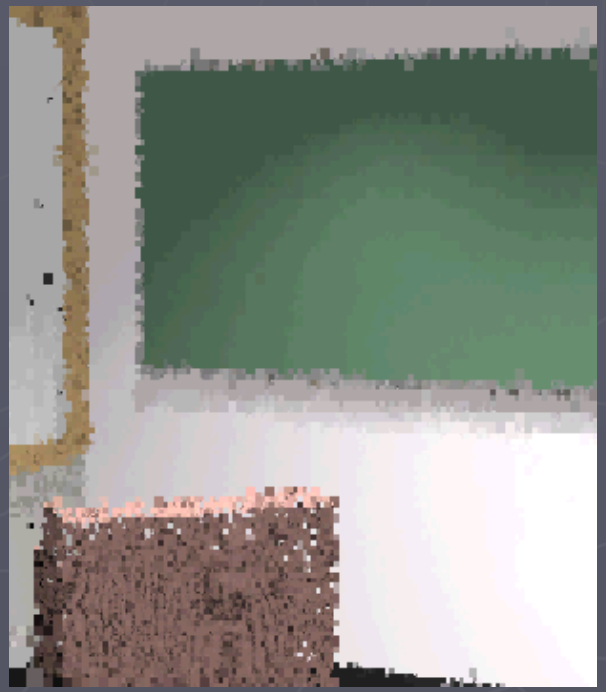
Quadtree Representation



One sample per leaf

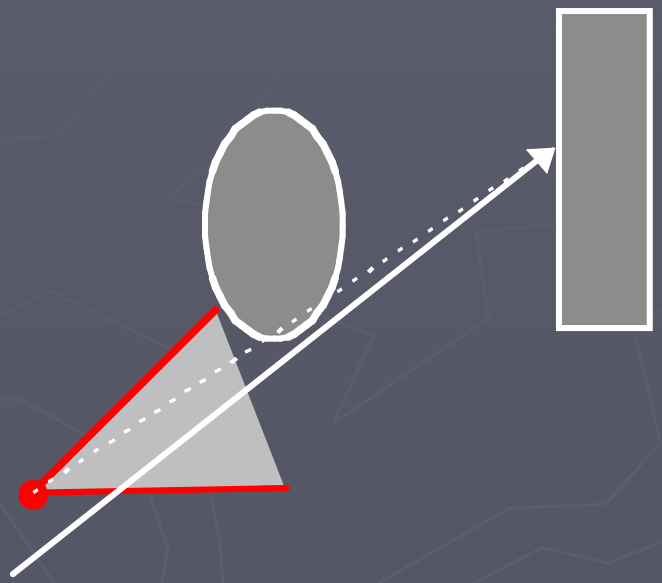


No filling



Average filling

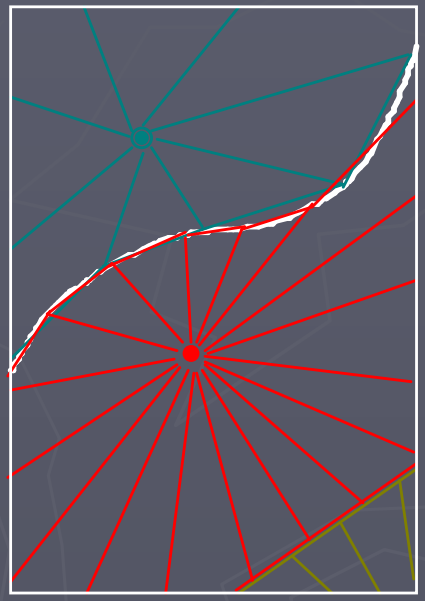
Multidepth Samples



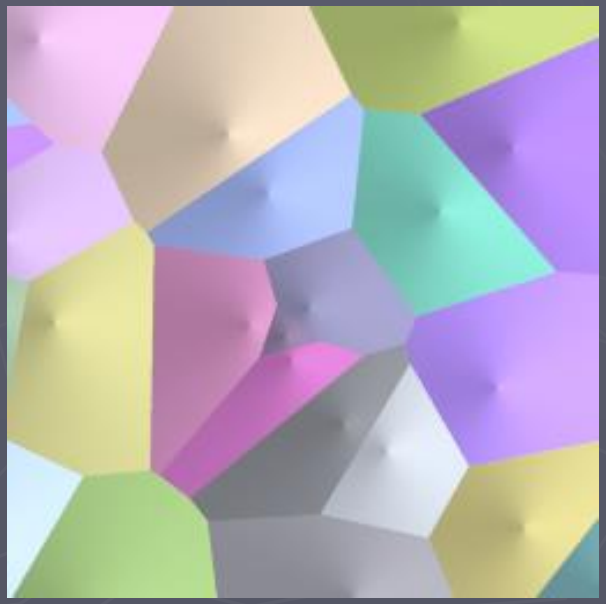
Because rays do not pass exactly through view point, multidepth samples result after reprojection to avoid loss of image focus. Our display method must attend to this.

Voronoi Cell Representation

Draw local geometry with OpenGL

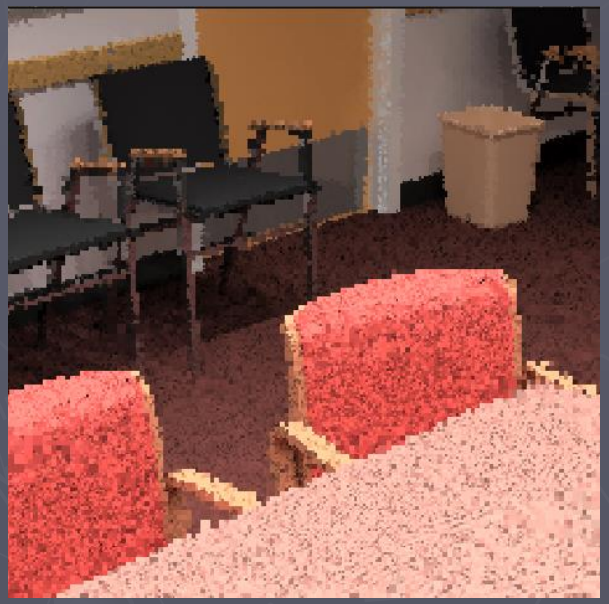


Constrain cones along depth discontinuities



Cones seen from above create Voronoi cells

Driver Comparison



Quadtree representation

Voronoi representation

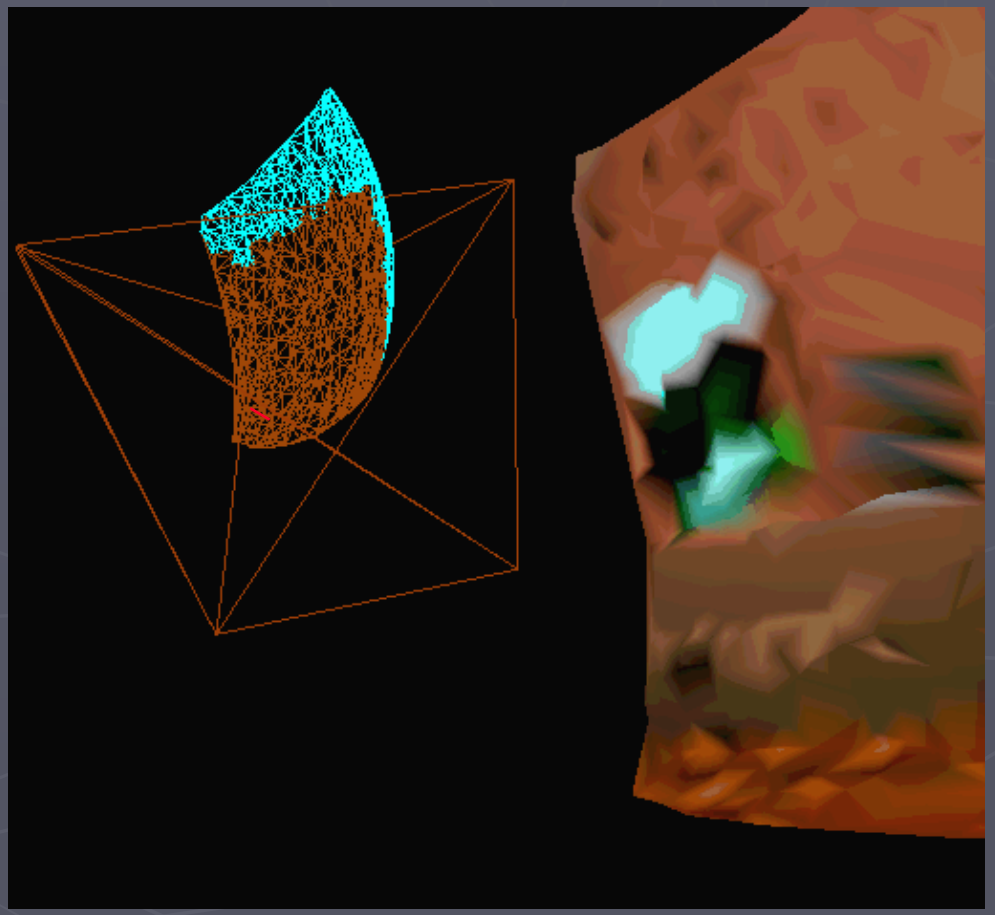
Mesh representation

Mesh Representation

- Maintain Delaunay triangulation of sphere centered on current eye point
  - each mesh vertex is a ray sample
  - vertex samples are added dynamically
  - view rotation uses the same mesh
  - viewpoint change may result in new mesh
- Mesh triangles are Gouraud shaded
  - new triangles are drawn over old ones

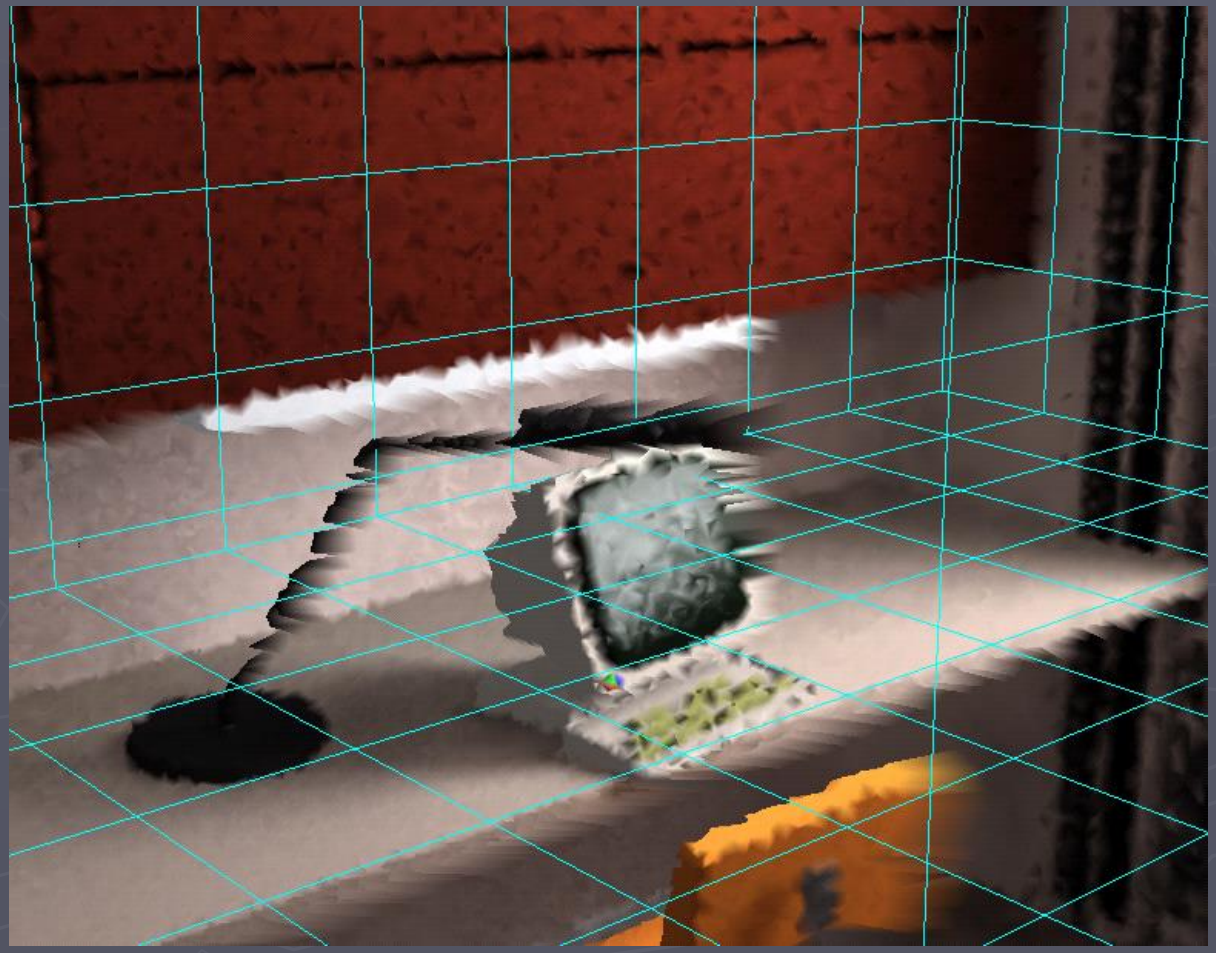
Mesh Representation (2)

- Samples stored in spherical mesh around eye point
  - spherical quadtree used for vertex location and frustum culling
  - fast vertex insertion is key



Example Mesh Rendering

From earlier motion sequence:



Dynamic Tone Mapping

- Quickly map dynamic range of scene samples into dynamic range of display
  - map every sample; update on redraw
- Optionally match human visibility
  - make visible on display what is visible in real life
- Use [Larson et al] from TVCG ‘97
  - histogram adjustment method

Tone Mapping Example

Linear

Camera model

Human model

Holodeck Programs

rholo - basic interface for interactive and batch holodeck rendering

rhinfo - extract holodeck header and section information and report fragmentation

rhcopy - copy ray samples to/from holodeck

rhoptimize - optimize (defragment) holodeck

rhpict - generate picture from holodeck

Cabin Example

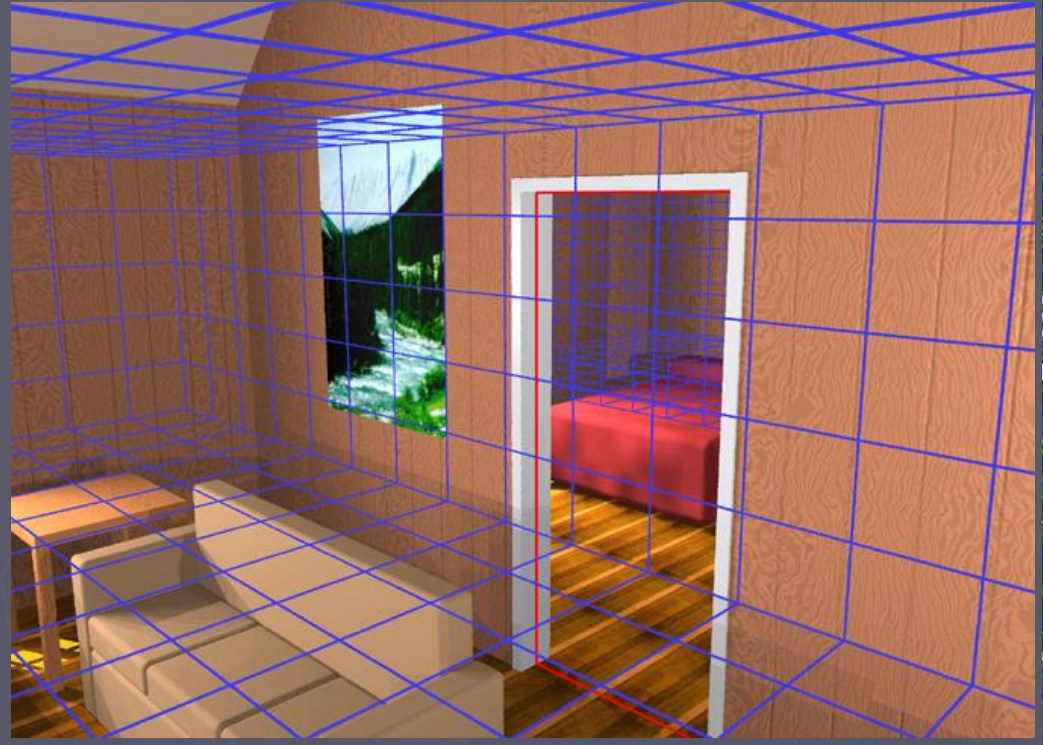
- Three holodeck sections for three rooms
  - Living room, bedroom, bathroom
  - Local geometry for each
- Holodeck input file:

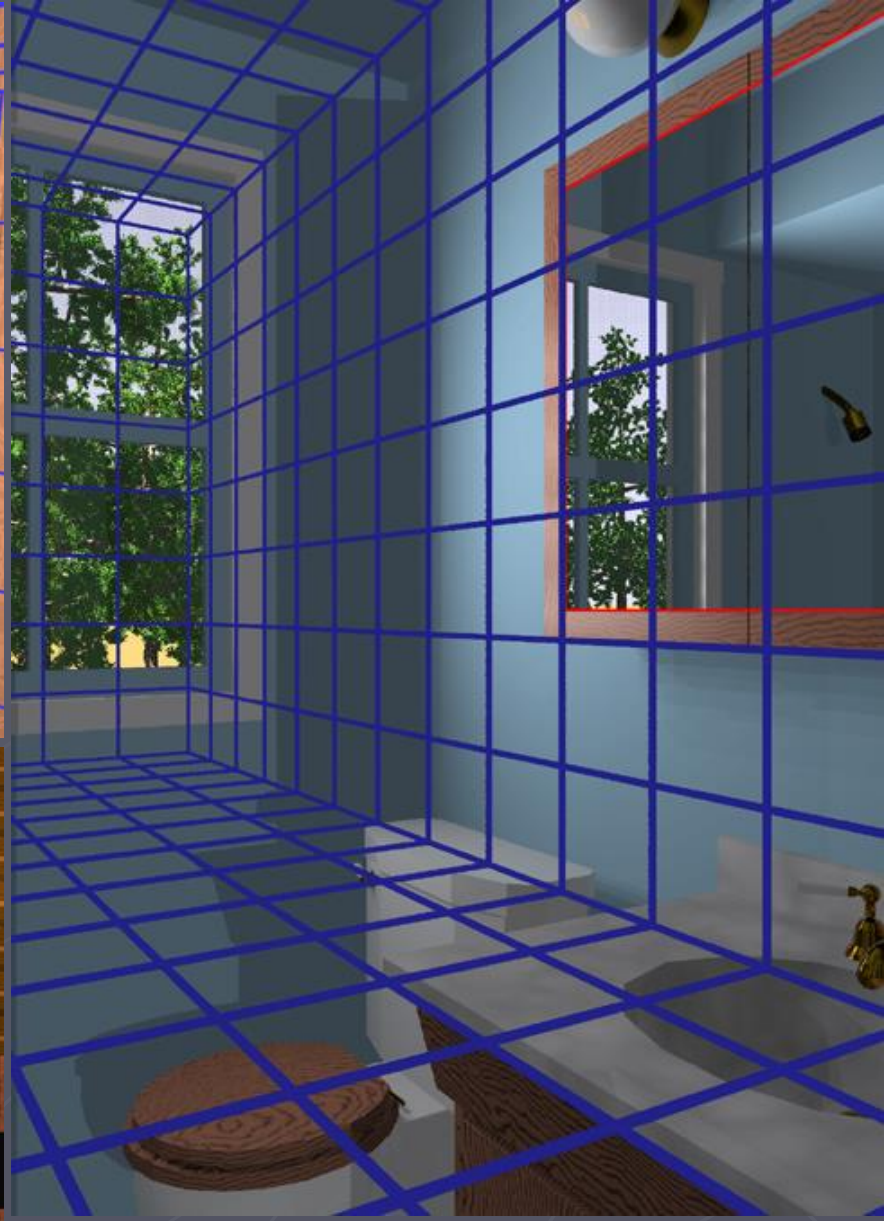
rad input file starting point

```
RIF= insummer.rif
section= 14 1 3 10 0 0 0 14 0 0 0 4
section= 1 1 3 8 0 0 0 8 0 0 0 4
section= .5 14 3 7 0 0 0 2 0 0 0 3
```

Holodeck sections

The specification above is all you really need.





Holodeck Grids

Portals

Some Other Holodeck Variables

OBSTRUCTIONS - T/F whether sections contain objects or not (default: unknown)

CACHE - amount of RAM to use for holodeck

DISKSPACE - maximum holodeck file size

TIME - maximum holodeck rendering time

Running **rholo**

Create empty holodeck from input file:

```
% rholo -f summer.hdk summer.hif
```

Render holodeck in background on 2 CPUs:

```
% rholo -n 2 summer.hdk &
```

Render holodeck interactively using OGL:

```
% rholo -n 1 -o ogl summer.hdk &
```

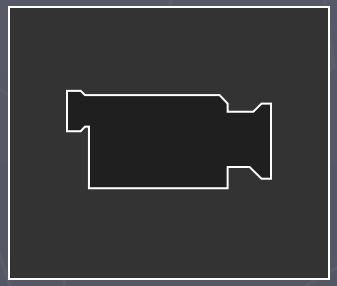
Control holodeck display from console:

```
% rholo -i -o x11 summer.hdk
```

Interactive Driver Commands

(mouse)	- view: approach, retreat, rotate, pivot
‘l’	- previous view
‘v’	- output current view to stdout
‘h’/’H’	- fix/release head height
^R	- redraw (recomputing tone-mapping)
^L	- reload samples from holodeck server
‘p’	- pause ray calculation
<cr>	- resume ray calculation
‘K’	- kill ray calculation
‘R’	- restart ray calculation
‘C’	- clobber holodeck contents (needs -f)
‘q’	- quit

Holodeck Demo



Conclusion on Holodeck

- New method for interactive ray tracing
- Multiprocessor ray calculation
- Dynamic ray caching
- Intermediate representation for display
- Facilitates continuous movement
- Applications in visual simulation, architecture, virtual reality

II. The *mesh* Primitive

- Efficient storage and rendering of large triangle mesh surfaces
  - Shared vertex information
- Vertices may hold local surface normal and (u,v) coordinates
  - Only way to get local coordinates in *Radiance*
- Syntax and semantics follow *instance* primitive

Using the *mesh* Primitive

- Compile .OBJ into *Radiance Triangle Mesh*

  % obj2mesh -a mats.rad tmsh.obj > tmsh.rtm

- Instantiate it in a *Radiance* scene file

  ```
  void mesh MyMesh1
    3 tmsh.rtm -rx 90
    0
    0
  ```

- Compile and render the scene

  % oconv scene.rad > scene.oct

  % rpict [options] scene.oct > scene.pic

Radiance 3.4 **gensurf** Surface

```
# gensurf mat OldMesh sin(s) t-cos(s) -t*t 3 3 -s

mat texfunc Phong
4 surf_dx surf_dy surf_dz surf.cal
0
13                  2
   -1.72942317   -0.0912916802    -0.13693752
   -1.63423044    -0.36621372   -0.0446389256
             0     1.66410059    -0.503849117
  0.0531230004     1.34831934    -0.451192852
0

Phong polygon OldMesh.1
0
0
12
   0.327194697    -0.944956946               0
   0.327194697    -0.611623613    -0.111111111
             0    -0.666666667    -0.111111111
             0              -1               0

mat texfunc Phong
4 surf_dx surf_dy surf_dz surf.cal
0
13                  2
  -0.718206438    -0.174896628   -0.0627037111
   -1.01574641    -0.261199779    0.0331146162
             0     0.735899411    -0.696150883
   0.118121819     0.343143398     -0.3342067

…etc.
```

one triangle, no (u,v)

Each vertex given ~5 times

New **gensurf** -o for .OBJ Output

```
# gensurf mat NewMesh sin(s) t-cos(s) -t*t 3 3 -o -s

usemtl mat

v            0            -1             0
                  vn 0 0 1
                  vt 0 0
v            0   -0.666666667   -0.111111111
                  vn 0 0.554700196 0.832050294
                  vt 0 0.333333333
v            0   -0.333333333   -0.444444444
                  vn 0 0.8 0.6
                  vt 0 0.666666667
v            0             0            -1
                  vn 0 0.894427191 0.447213595
                  vt 0 1
v    0.327194697   -0.944956946             0
                  vn 0 0 1
                  vt 0.333333333 0
v    0.327194697   -0.611623613   -0.111111111
                  vn -0.188619363 0.544743478 0.817115218
                  vt 0.333333333 0.333333333
v    0.327194697    -0.27829028   -0.444444444
                  vn -0.266950475 0.770968199 0.578226149
                  vt 0.333333333 0.666666667
v    0.327194697   0.0550430537            -1
                  vn -0.295836084 0.854391485 0.427195742
                  vt 0.333333333 1
f 5/5/5 6/6/6 2/2/2 1/1/1
f 2/2/2 6/6/6 7/7/7 3/3/3
f 7/7/7 8/8/8 4/4/4 3/3/3

…etc.
```

three triangles, with (u,v)

Each vertex specified only once, providing connectivity information needed by *mesh*

Rendering Comparison 1

1 Million Triangle Sculpture
(no surface smoothing)

Model courtesy Paul Debevec, ICT

	instance	*mesh*
Compile scene	94 sec 400 Meg	60 sec 400 Meg
Render picture	32 sec 208 Meg	34 sec 61 Meg

Rendering Comparison 2

500K Triangle Surface
(w/ normal smoothing)

	instance	*mesh*
Compile scene	39 sec 274 Meg	30 sec 269 Meg
Render picture	18 sec 208 Meg	10 sec 36 Meg

Significant savings from not calling .cal routines during surface normal evaluation

Rendering Comparison 3

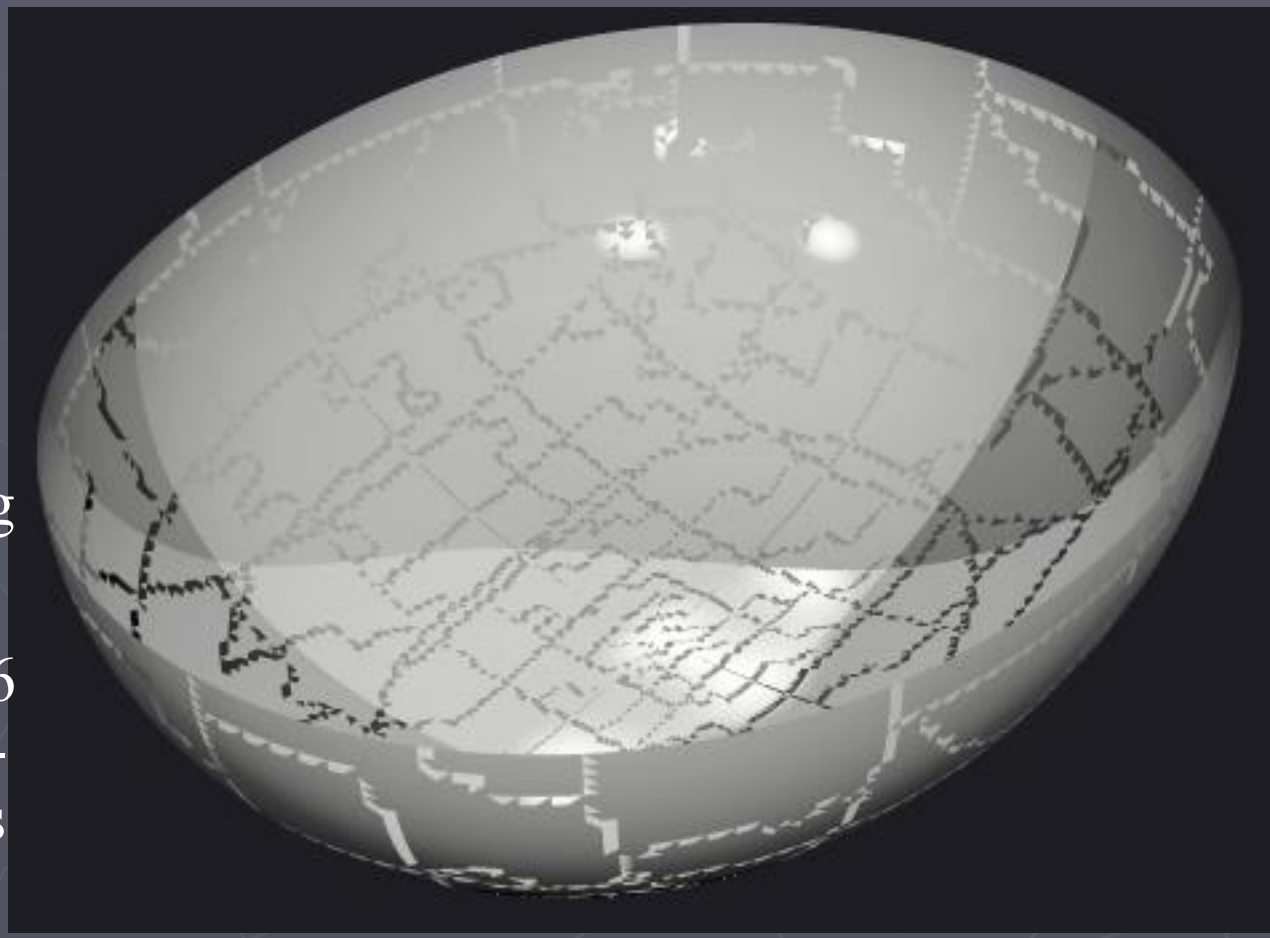
140K Triangle Chair Model
(w/ normals & local coords.)
Model courtesy Bruce Carlin, DecorMagic

	instance	*mesh*
Compile scene	9 sec 78 Meg	7 sec 78 Meg
Render picture	13 min 208 Meg	16 min 36 Meg

Important benefits of local (u,v)

How We Save Memory

- Vertices use 4-byte integer per dimension (12) and local coordinates use 2 bytes per dim. (4)
- Normals are packed into a 4-byte encoding
- Mesh is broken into patches of at most 256 vertices, permitting 1-byte vertex references



An early test image showing patch boundaries

Mesh Storage Requirements

Storage/triangle

Vertex type	*Radiance* 3.4	*mesh* primitive
simple	146 bytes	< 16 bytes
+ normals	+130 bytes	+4 bytes
+ (u,v)	n/a	+4 bytes

Can hold mesh with 9 times as many triangles in the same amount of memory

Mesh Problems in 3.5 Release

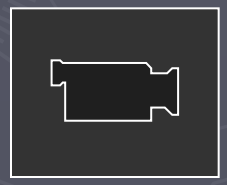
- Local (u,v) coordinates broken
- *mesh* instances aren't properly freed
- **xform** doesn't add transforms to mesh instances as it should
- All of the above is fixed in current HEAD distribution available from www.radiance-online.org

Conclusion on *mesh* Primitive

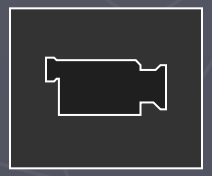
- As easy to use as *instance* primitive
- Provides local (u,v) coordinates to map patterns
- Saves up to 85% of rendering memory
- **obj2mesh** program compiles .OBJ files
  - .OBJ is simplest vertex-sharing format
  - New 3.5 **gensurf** can produce .OBJ directly
  - Many CAD programs export .OBJ as well
  - Future support for MGF?

III. The **ranimove** Program

- "Just in time" animation system
  - You tell it when to finish and it does its best
- Exploits inattentional blindness and IBR
  - WYDSYWM: What you don't see you won't miss
  - Image-based motion blur and sample extrapolation accelerates rendering
- Error visibility tied to attention and motion



Detail2Attention

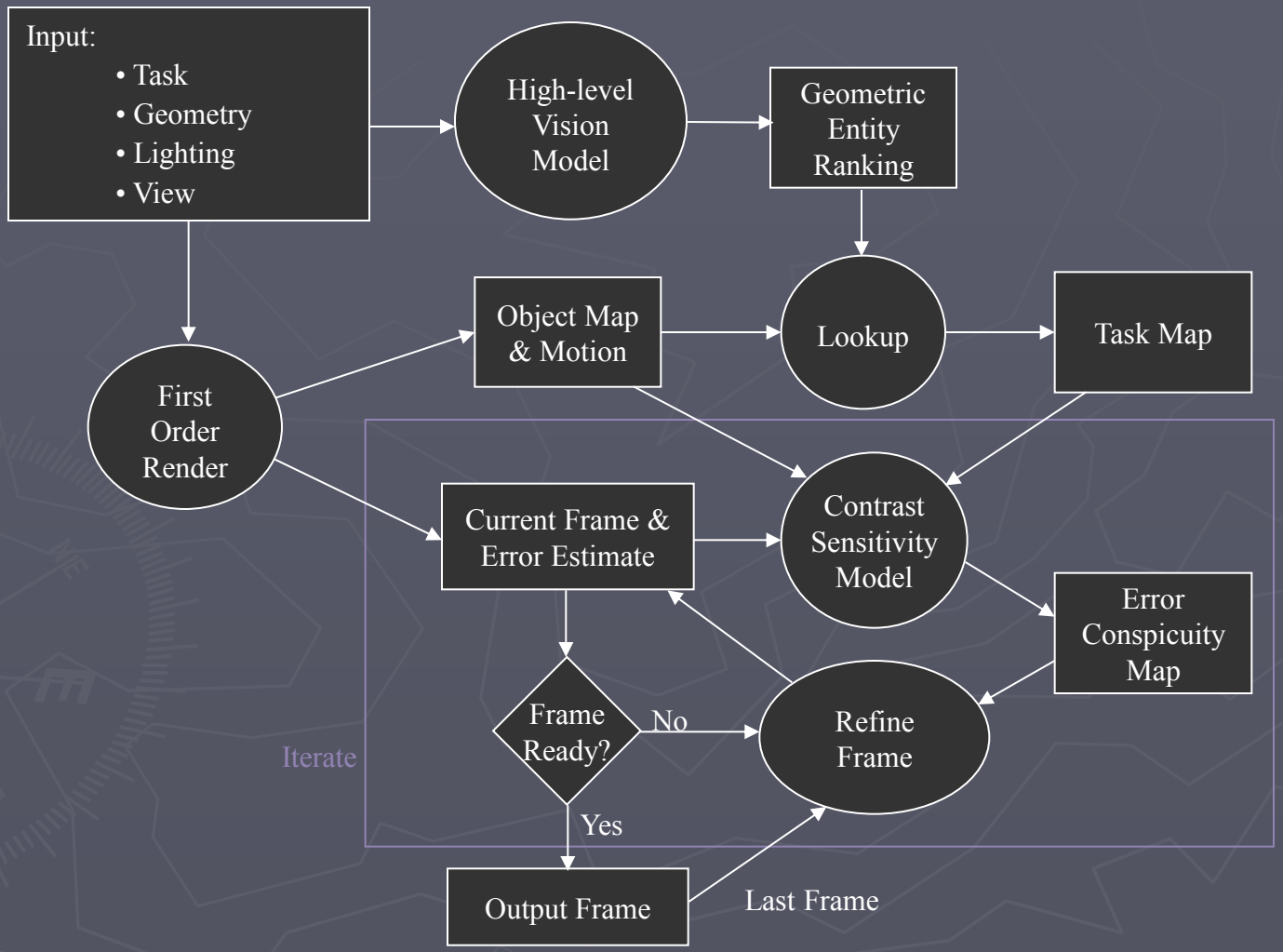


DVD

ranimate vs. **ranimove**

	ranimate	**ranimove**
Execution mode	Runs **rpict**, **pinterp** & **pfilt**	Calls rendering library
Network rendering		
File management		
Object motion blur		
Progressive		

WYDSYWM Framework



Example **ranimove** Input

```
#
# First animation segment input for ranimove
#
RIF= shipfull.rif
VIEW= seg1all.vf
RESOLUTION= 640 480 0
RATE= 15
MBLUR= .35
BASENAME= seg1/s1_%04d
#
# Moving objects
#
# move= parent this transform radfile [priority]
move= void TV seg1TVall.xf TV.rad 1.5
move= TV topRotor topTVrotor.xf topTVrotor.rad .5
move= TV botRotor botTVrotor.xf botTVrotor.rad .5
move= TV screen . "!genTVscreen.csh 90.03 15" 1.5
move= TV strobe . "!genTVstrobe.csh 90.03 15"
move= void lanterns . blant.rad 2.5
move= void extinguishers . exting.rad 2
```

Target frame rate

Frame fraction for virtual shutter

ranimove manages object motion based on the specification below

The optional priority value indicates the importance of this object relative to others in the scene

Example **ranimove** Input

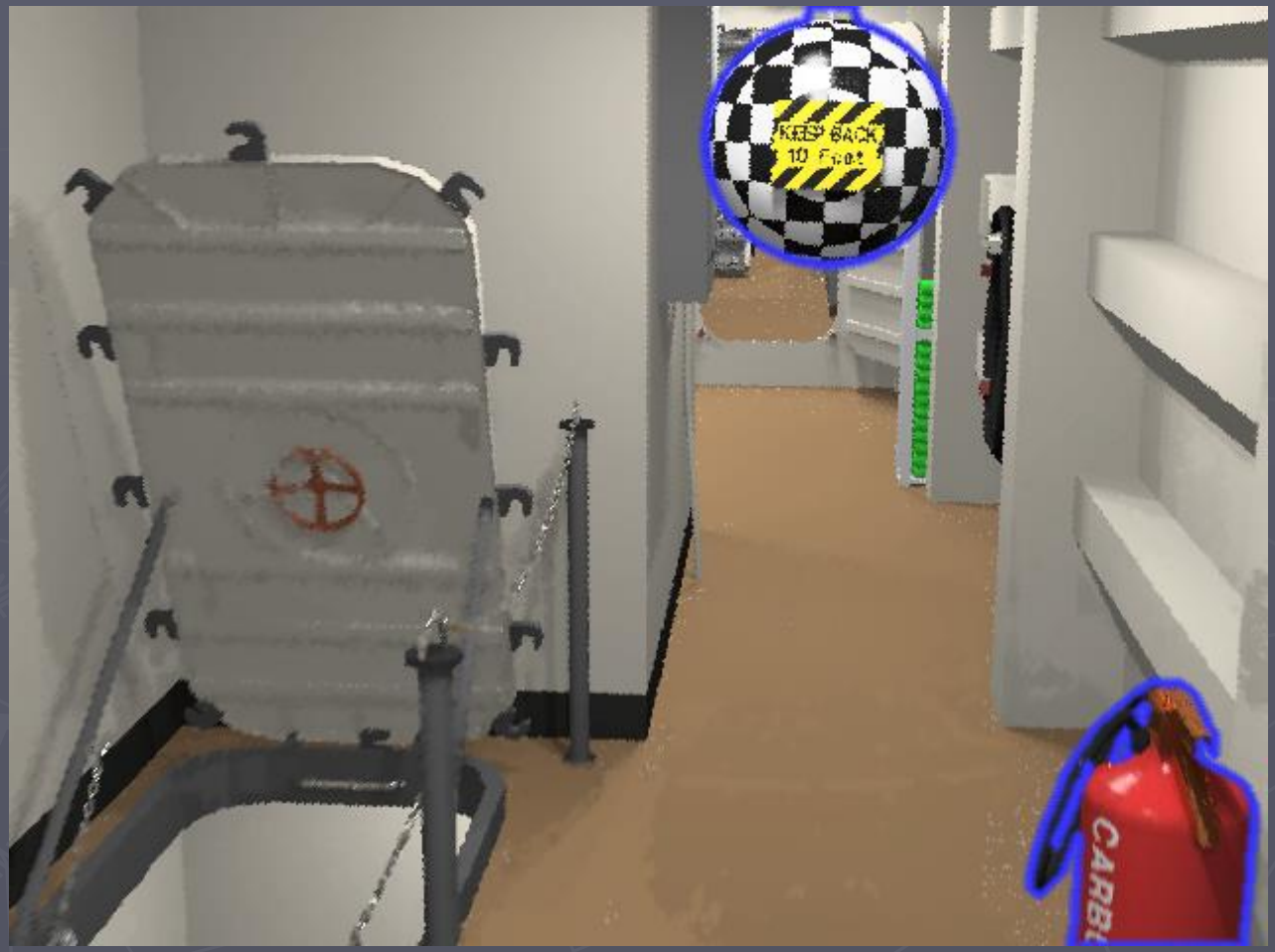
```
#
# First animation segment input
#
RIF= shipfull.rif
VIEW= seg1all.vf
RESOLUTION= 640 480 0
RATE= 15
MBLUR= .35
BASENAME= seg1/s1_%04d
#
# Moving objects
#
# move= parent this transform radfile [r
move= void TV seg1TVall.xf TV.rad 1.5
move= TV topRotor topTVrotor.xf topTV
move= TV botRotor botTVrotor.xf botTV
move= TV screen . "!genTVscreen.csh 90.03 15" 1.
move= TV strobe . "!genTVstrobe.csh 90.03 15"
move= void lanterns . blant.rad 2.5
move= void extinguishers . exting.rad 2
```

```
VIEW= -vtv -vh 40 -vv 30.537 -
VIEW= -vp  78.9133  23.9897
VIEW= -vp   78.816   23.978
VIEW= -vp  78.7086  23.9651
VIEW= -vp  78.5917  23.9511
VIEW= -vp  78.4657  23.9359
VIEW= -vp  78.3312  23.9198
VIEW= -vp  78.1887  23.9026
VIEW= -vp  78.0387  23.8845

…etc.
```

```
-rx 2.00026 -ry 0.2
-rx 1.94718 -ry 0.2
-rx 1.88744 -ry 0.2
-rx 1.82145 -ry 0.3
-rx 1.74964 -ry 0.3
-rx 1.67242 -ry 0.3
-rx 1.59022 -ry 0.4
-rx 1.50345 -ry 0.4
…etc.
```

```
#!/bin/csh -f
#
# Generate nth frame of our narration
#

set frm=`ev "ceil(($1+$3/$2)*15)"`

if ($frm < 10) then
        set i=000$frm
else if ($frm < 100) then
        set i=00$frm
else if ($frm < 1000) then
        set i=0$frm
else
        set i=$frm
endif

djpeg narration_jpeg/frame$i.jpg | ra_ppm -r \
                        > narrator.pic

cat << '_EOF_'
void colorpict screenpicture
7 red green blue narrator.pic
        . (Py*12+2.5)/3.75 (Pz*12+1.875)/3.75
0
0
screenpicture glow screenglow
0
0
4 .7 .7 .7 0

!(echo screenglow polygon screen 0 0 132 ; \
        rcalc -e '$1=.295;$2=$1/12;$3=$2/12' roundrect.pts)
'_EOF_'
```

Example Frame w/ Task Objects



Error Map Estimation

- Stochastic errors may be estimated from neighborhood samples
- Systematic error bounds may be estimated from knowledge of algorithm behavior
- Estimate accuracy is not critical for good performance

Initial Error Estimate

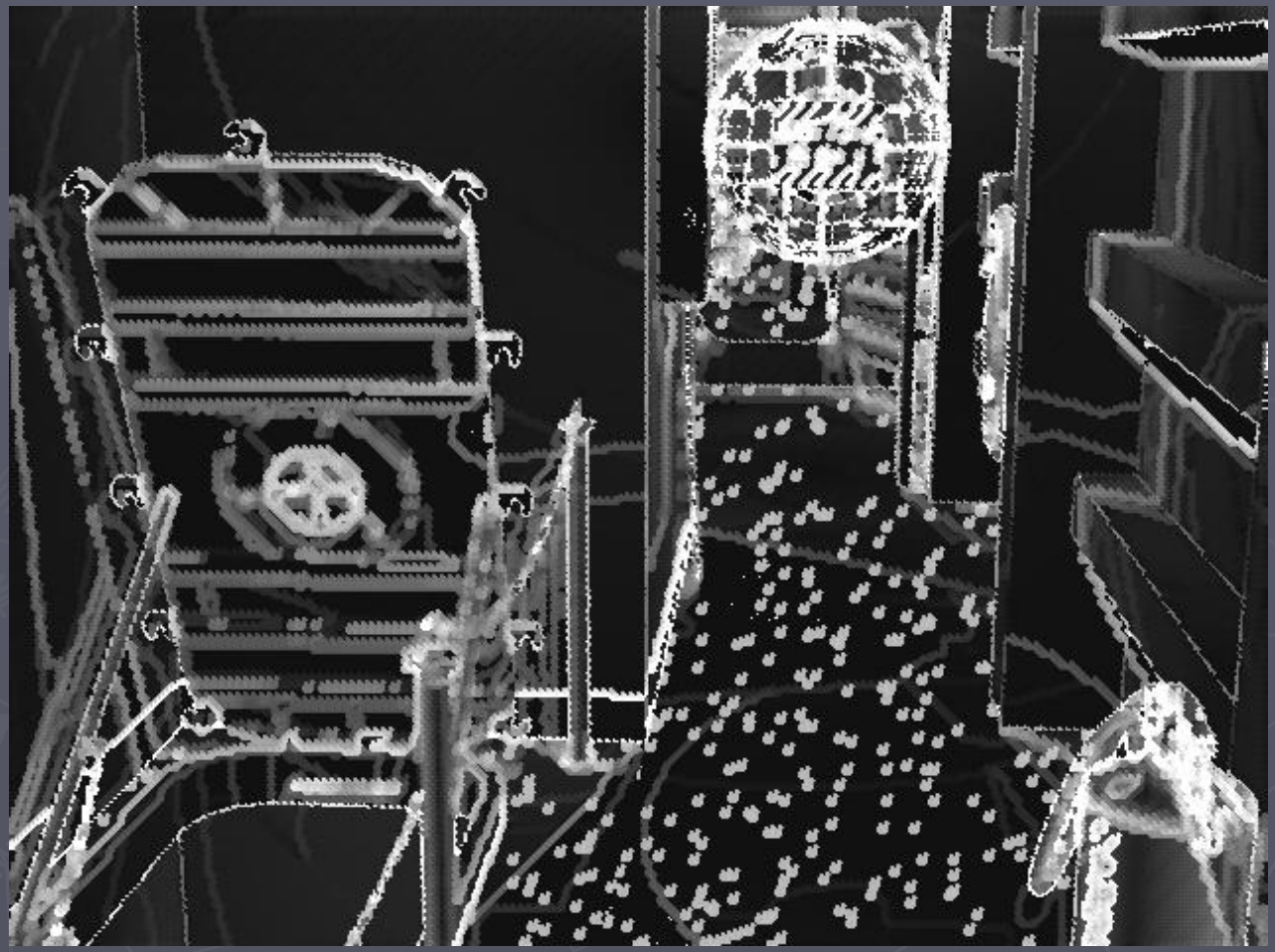


Image-based Refinement Pass

- Since we know exact motion, IBR works very well in this framework
- Select image values from previous frame
  - Criteria include coherence, accuracy, agreement
- Replace current sample and degrade error
  - Error degradation results in sample retirement

Contrast Sensitivity Model

Additional samples are directed based on Daly's CSF model:

$$CSF(\rho, v_R) = k \cdot c_0 \cdot c_2 \cdot v_R \cdot (c_1 2\pi\rho)^2 \exp\left(-\frac{c_1 4\pi\rho}{\rho_{\max}}\right)$$

where:

ρ is spatial frequency

v_R is retinal velocity

$$k = 6.1 + 7.3\left|\log(c_2 v_R / 3)\right|^3$$

$$\rho_{\max} = 45.9/(c_2 v_R + 2)$$

$c_0 = 1.14, \quad c_1 = 0.67, \quad c_2 = 1.7$ for CRT at 100 cd/m2

Error Conspicuity Model

Retinal velocity depends on task-level saliency:

$$v_R = \left| v_I - \min(v_I \cdot S / S_{\max} + v_{\min},\ v_{\max}) \right|$$

where:

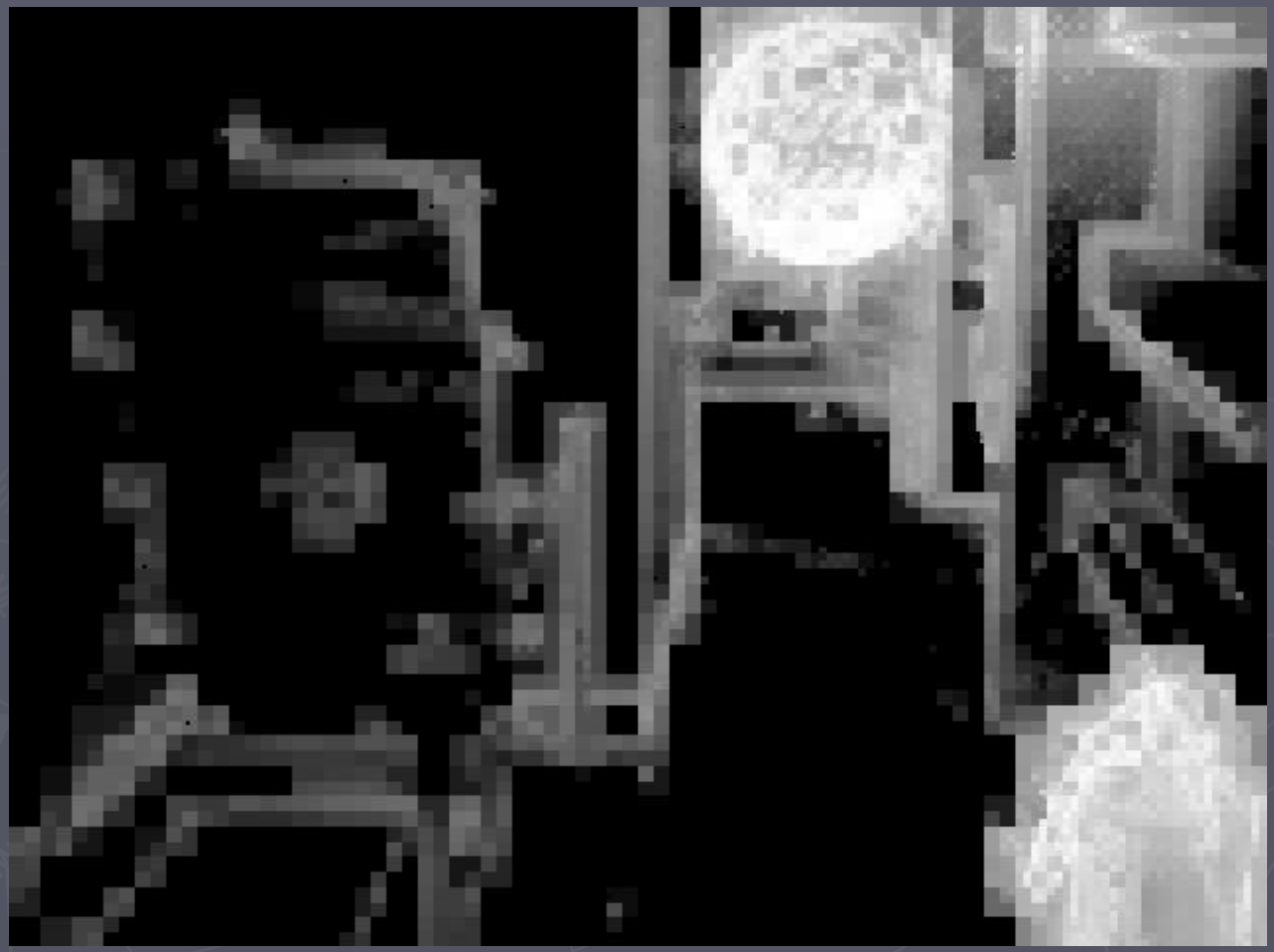
- v_I = local pixel velocity (from motion map)
- S = task-level saliency for this region
- $S_{\max}$ = max. saliency in this frame, but not less than 1/0.82
- $v_{\min}$ = 0.15°/sec (eye drift velocity)
- $v_{\max}$ = 80°/sec (movement-tracking limit)

$$EC = S \cdot \max(E \cdot CSF / ND - 1,\ 0)$$ *Error Conspicuity*

where:

- E = relative error estimate for this pixel
- ND = noticeable difference threshold

Error Conspicuity Map

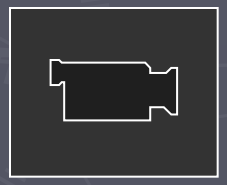


Implementation Example

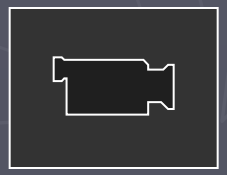
- Compared to a standard rendering that finished in the same time, **ranimove** produced better quality on task objects
- Rendering the same high quality over the entire frame would take about 7 times longer using the standard method

Example Animation

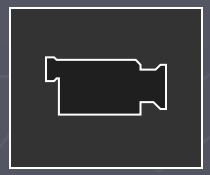
- The following animation was rendered at two minutes per frame on a 2000 model G3 laptop computer (long may it live)
- Many artifacts are intentionally visible, but less so if you are performing the task



Detail2Attention



Seg1Bonus



DVD

Conclusion on **ranimove**

- Better than **ranimate** when object motion and rendering time/quality trade-offs are more important than rendering management
- Takes a little effort to prepare the control file, but like the holodeck, it's usually worth it
- Network rendering can still be done, but the user has to coordinate frame segments

What to Expect in *Radiance* 3.6

- Fixes to *mesh* primitive (obviously)
- New -N option to rad to start multiple rendering processes
- Increase in maximum holodeck size
- Binary I/O in **rcalc** program
- 16-bit/sample and LZW support in **ra_tiff**
- Windows ready??