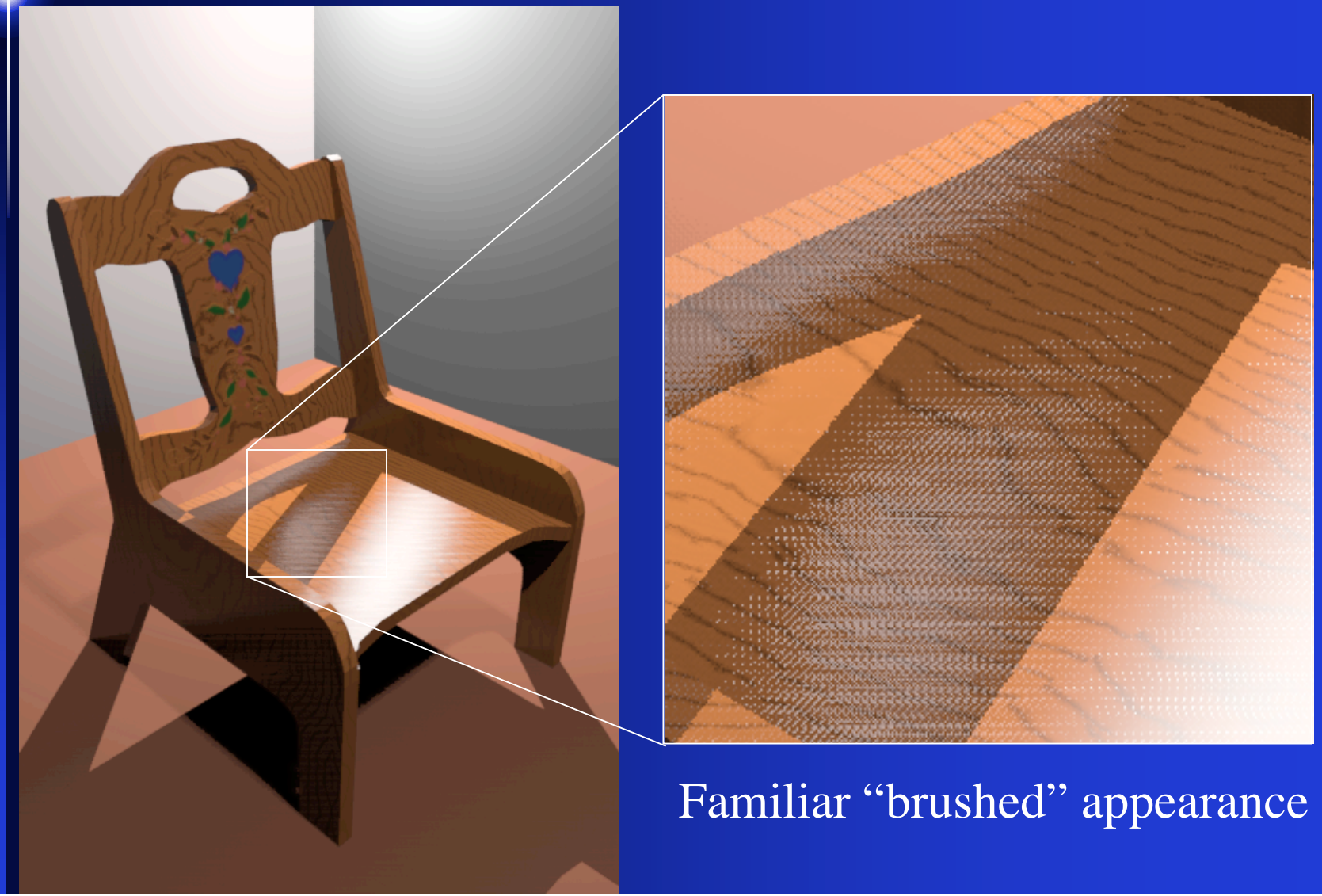
Code Changes and Additions in *Radiance* 3.7

Greg Ward
Anyhere Software

What's New in 3.7

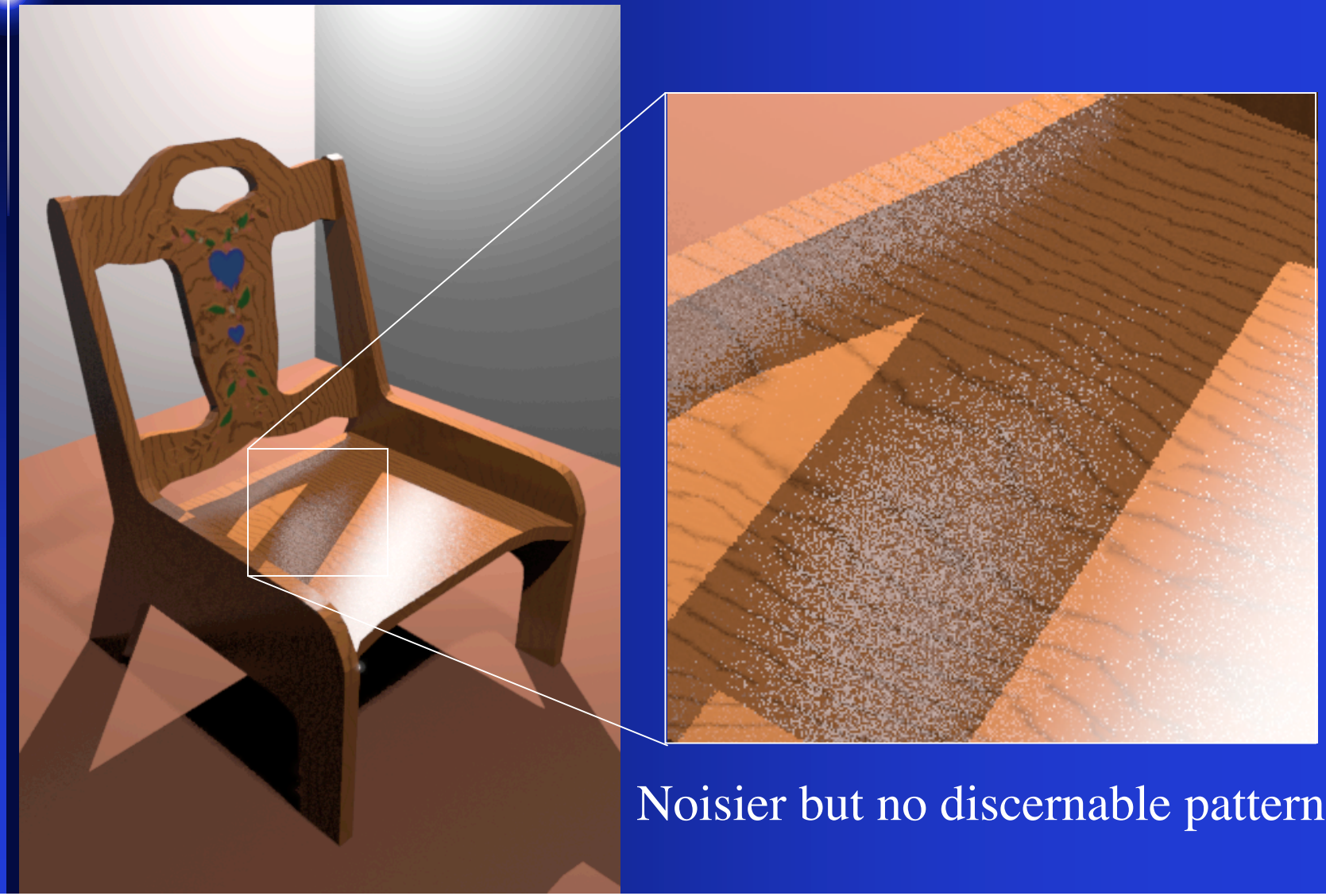
- Added pure Monte Carlo (`-u`) option
- Added depth-of-field (`-pd`) option
- Added "Russian roulette" ray termination
- Enhanced interreflection calculation
- **mksource** program to identify light sources
- **rtcontrib** program for contribution coeff.
- Created **meta2bmp** metafile converter
- Retired several unused programs

Quasi-Monte Carlo (–u–)



Familiar “brushed” appearance

Pure Monte Carlo (–u+)



Noisier but no discernable pattern

Depth-of-Field Sampling (`-pd`)

- Assigns aperture diameter in world units
- Also requires focus distance
  - Uses length of view direction (`-vd`) vector
  - Set by new **rvu** "focus" command
- More accurate than **pdfblur** with **pinterp**

Depth-of-Field Example

No blur

Depth-of-Field Example

pdfblur with **pinterp**
(32 samples)

Depth-of-Field Example

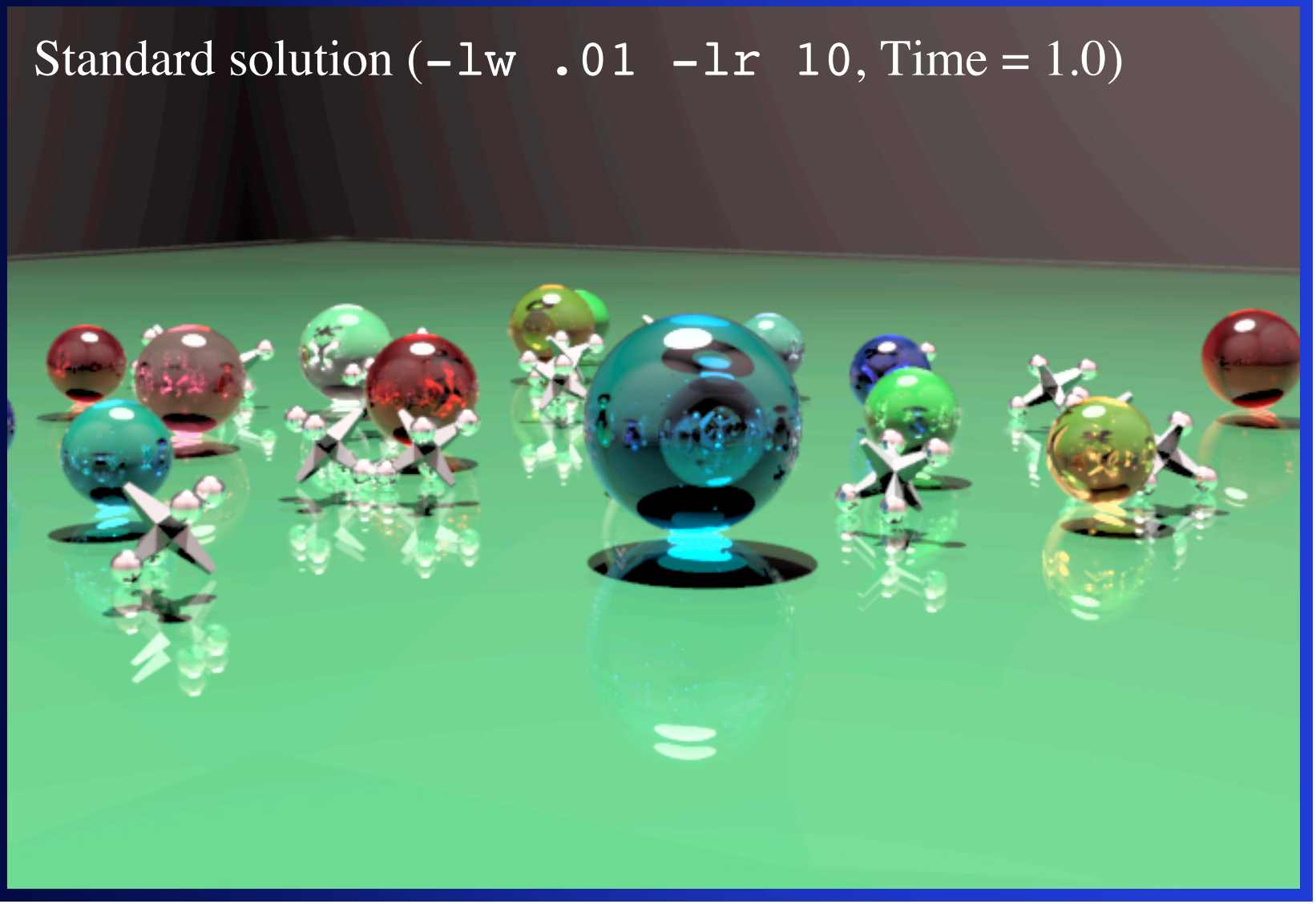
rpict with -pd option

Russian Roulette Sampling

- Standard algorithm is strict termination below `-lw` minimum weight limit
- Russian roulette continues trace with probability of (ray_weight/min_weight)
  - Continued ray weight is reset to min_weight
- Eliminates bias due to ray termination
- Enabled with negative value to `-lr` option
- Made possible by code modifications needed for **rtcontrib**

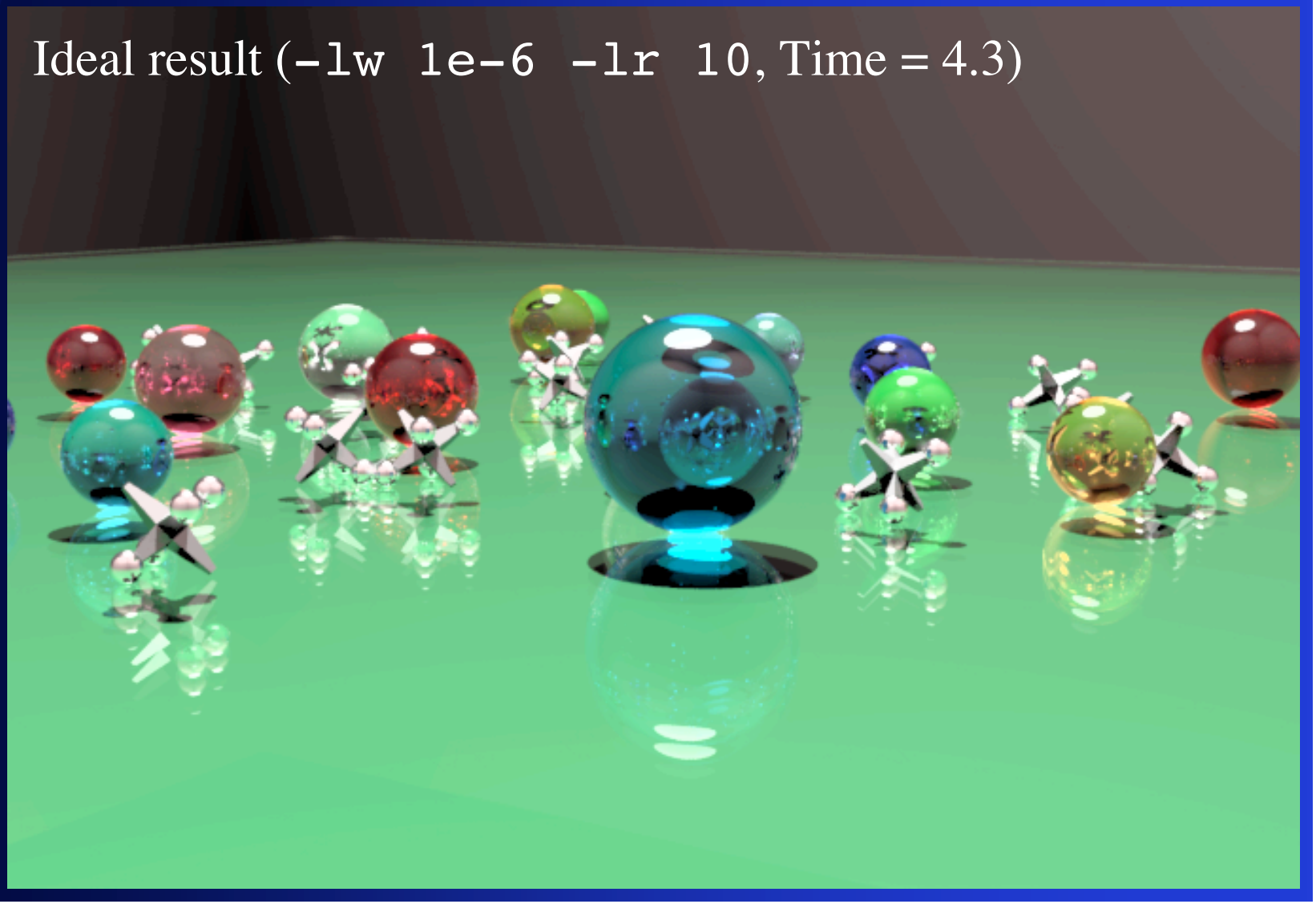
Russian Roulette Example

Standard solution (`-lw .01 -lr 10`, Time = 1.0)



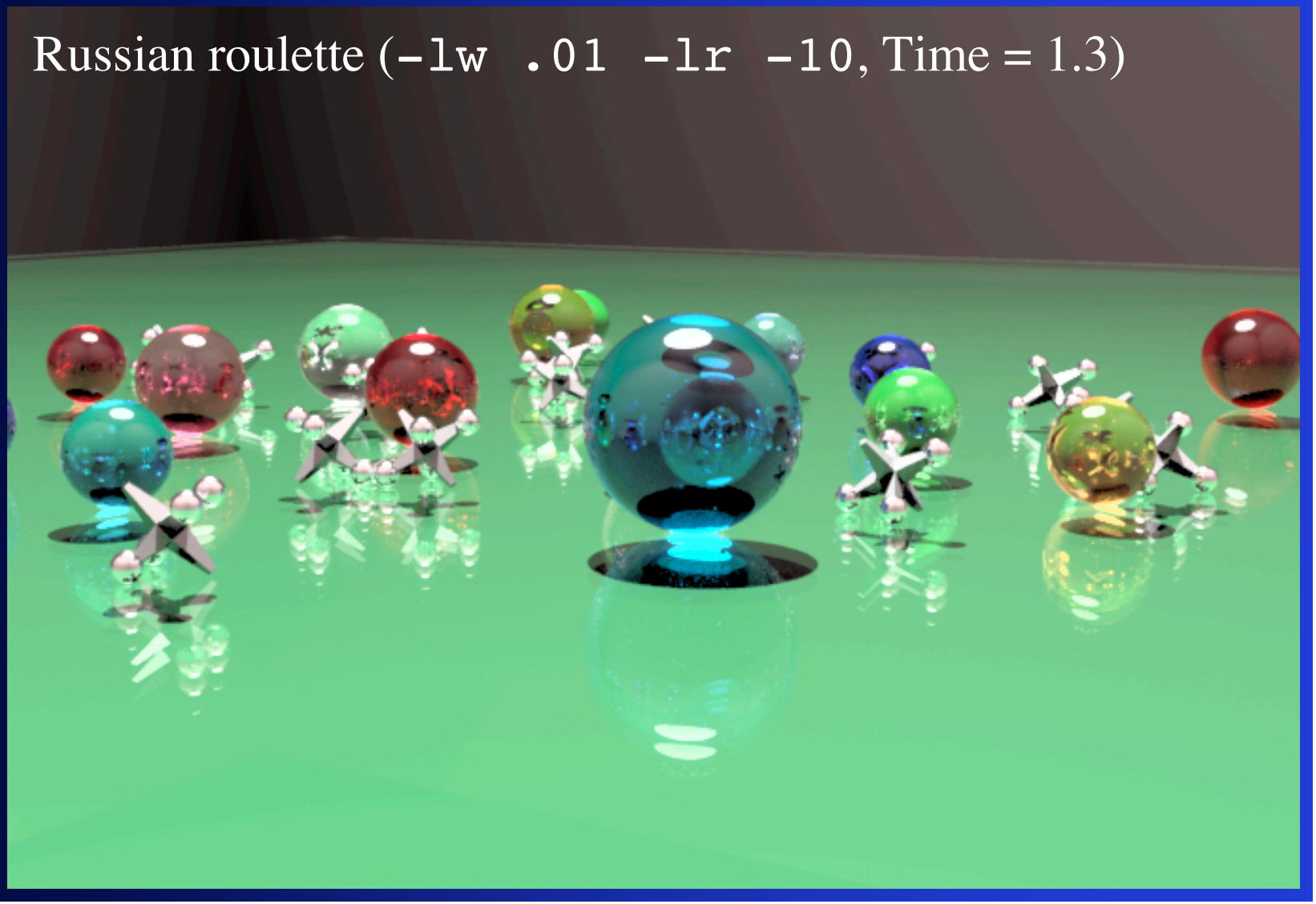
Russian Roulette Example

Ideal result (`-lw 1e-6 -lr 10`, Time = 4.3)



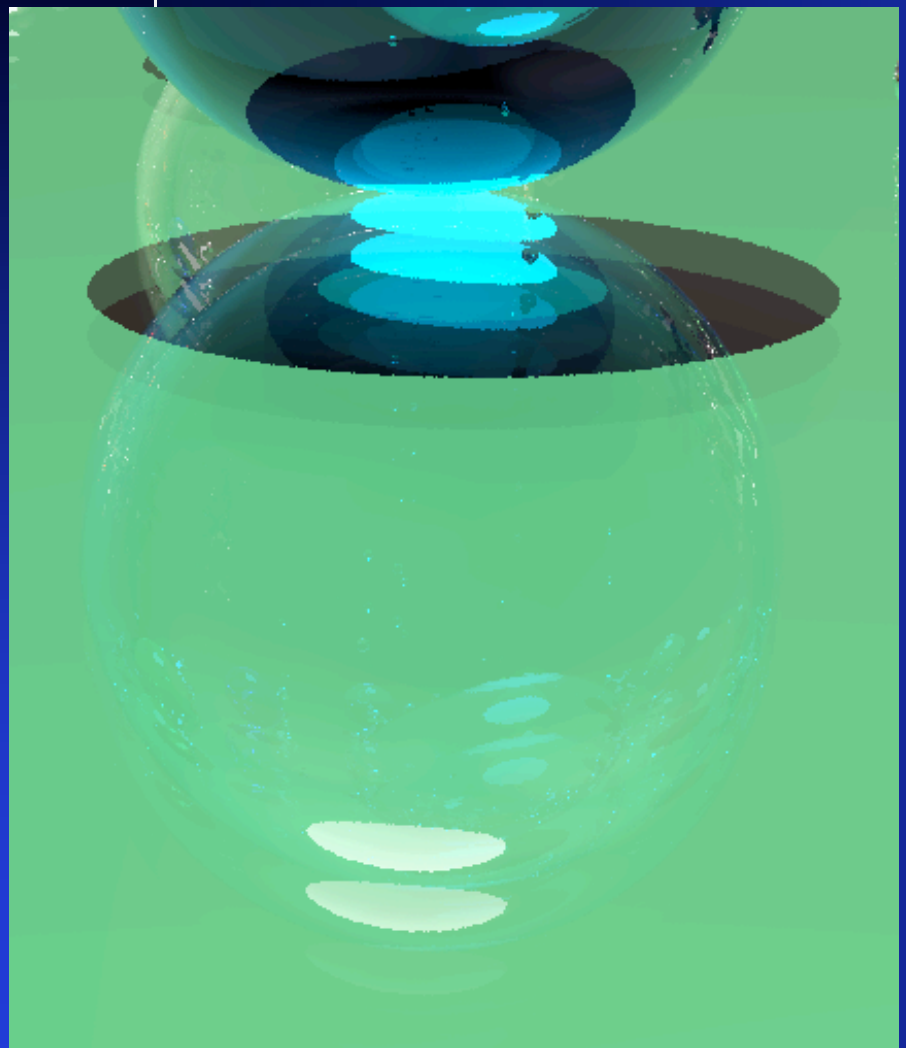
Russian Roulette Example

Russian roulette (`-lw .01 -lr -10`, Time = 1.3)

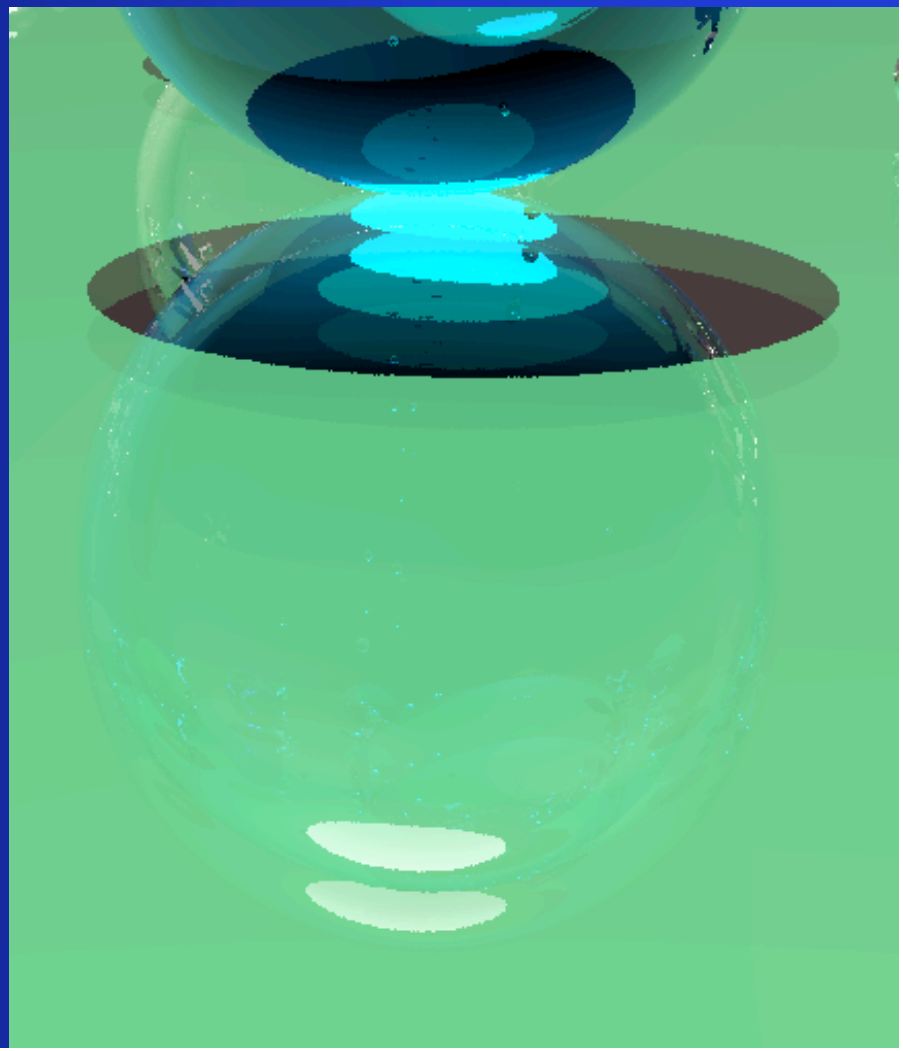


Russian Roulette Example

Ideal result

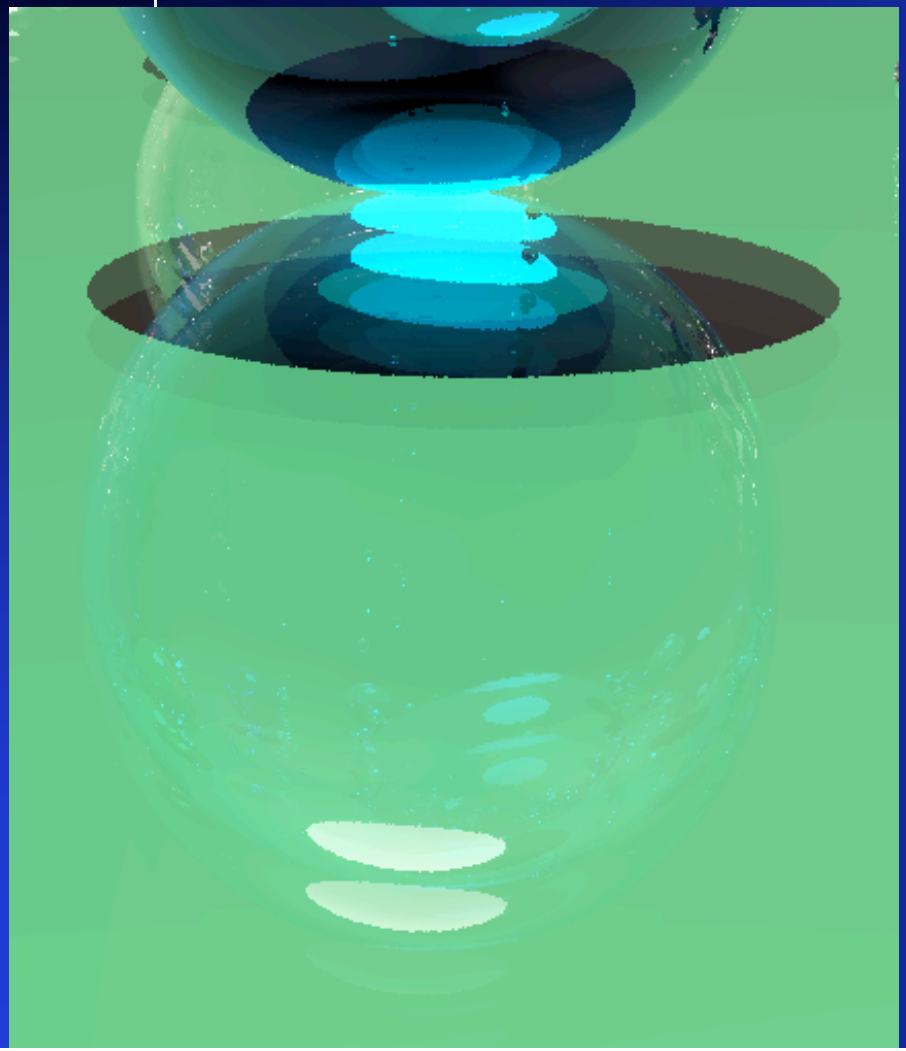


Standard Solution

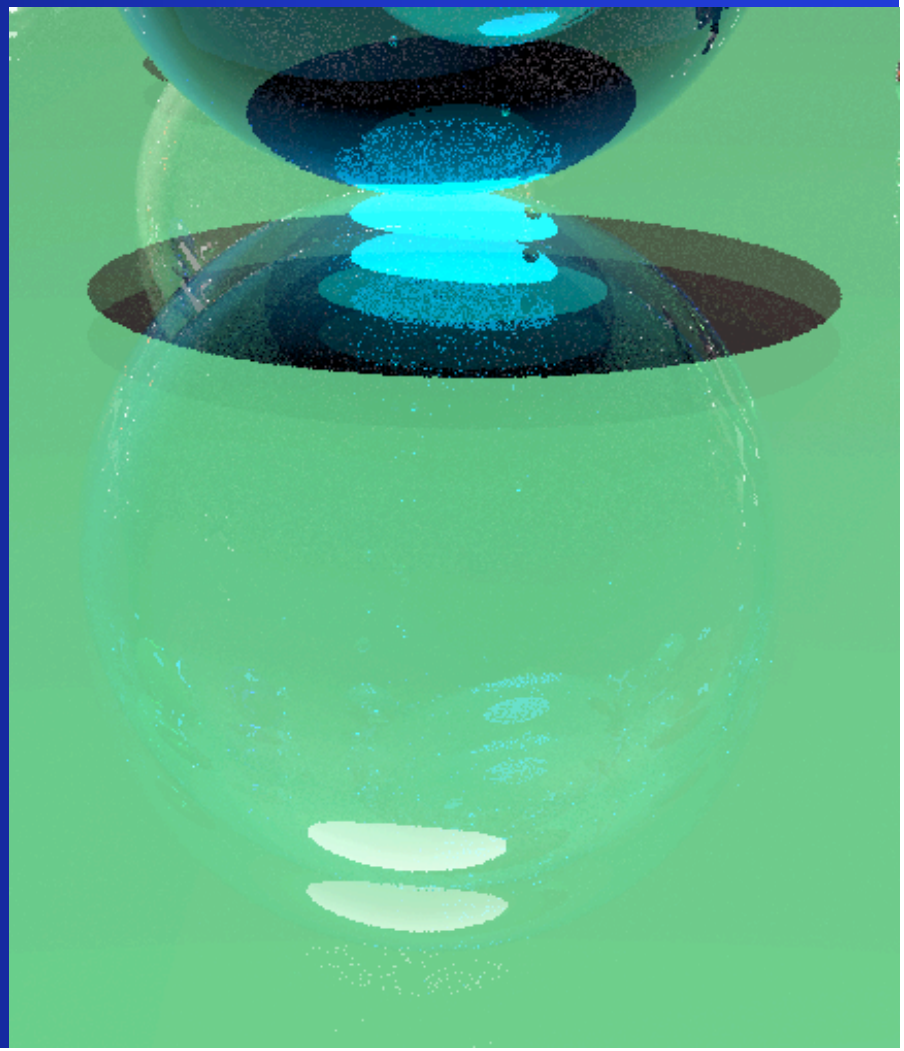


Russian Roulette Example

Ideal result



Russian roulette



mksource Program

- Identifies bright regions in HDR environment map (*light probe*)
- Superimposes distant *illum source*'s
- Reduces noise in resulting rendeirngs
- Only works with *glow source* inputs

mksource Algorithm

- Sample all directions using geodesic mesh
- Determine threshold from top 2%
- Loop until no pixels over threshold:
  1. Identify brightest unclaimed pixel
  2. Grow source towards brightest unclaimed perimeter until:
     a) Source exceeds maximum size, or
     b) Perimeter values all below threshold, or
     c) Source average drops below threshold
- + Methods to avoid over- and under-counting

Example mksource Results

Example mksource Results

Example mksource Results

rtcontrib Program

- Exploits new capability in **rtrace** for reporting ray contribution coefficients
- Contribution coefficients only useful deep in ray tree ⇒100+ million values per image
- **rtcontrib** sums & tabulates coefficients
  - Flexible control over where to gather rays
  - Output can be *Radiance* picture files, ASCII tables, or binary float's or double's
- **pcomb** useful in summing coeff. pictures

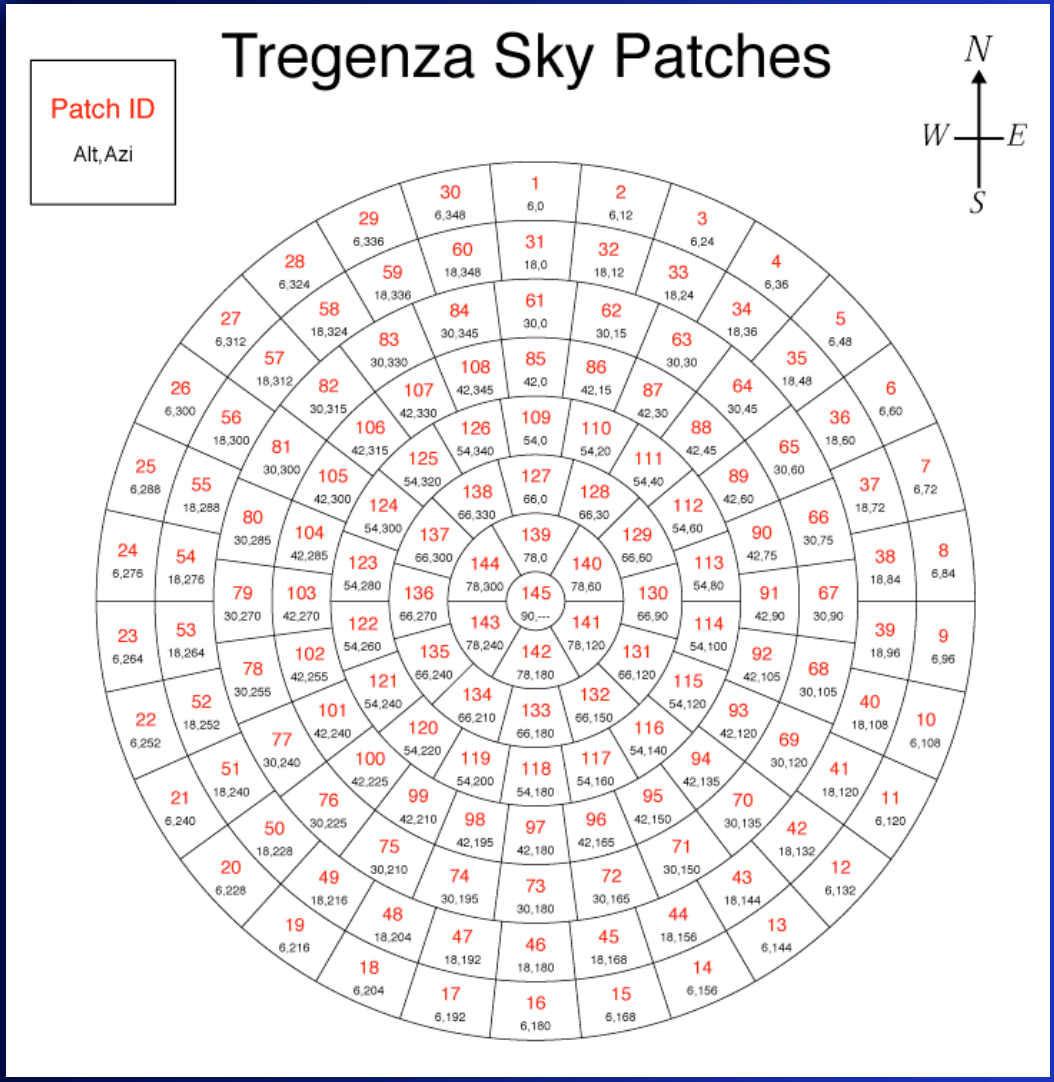
Simple rtcontrib Example

% `rtcontrib @render.opt -I+ -o c_%s.dat -m light1 -m light2 scene.oct < test.dat`

- Produces two files: c_light1.dat & c_light2.dat
- These contain irradiance coefficients for our two sources

- Linear combination of coefficients yields illuminance for any dimming settings
- Advantage: 1 calculation rather than 2

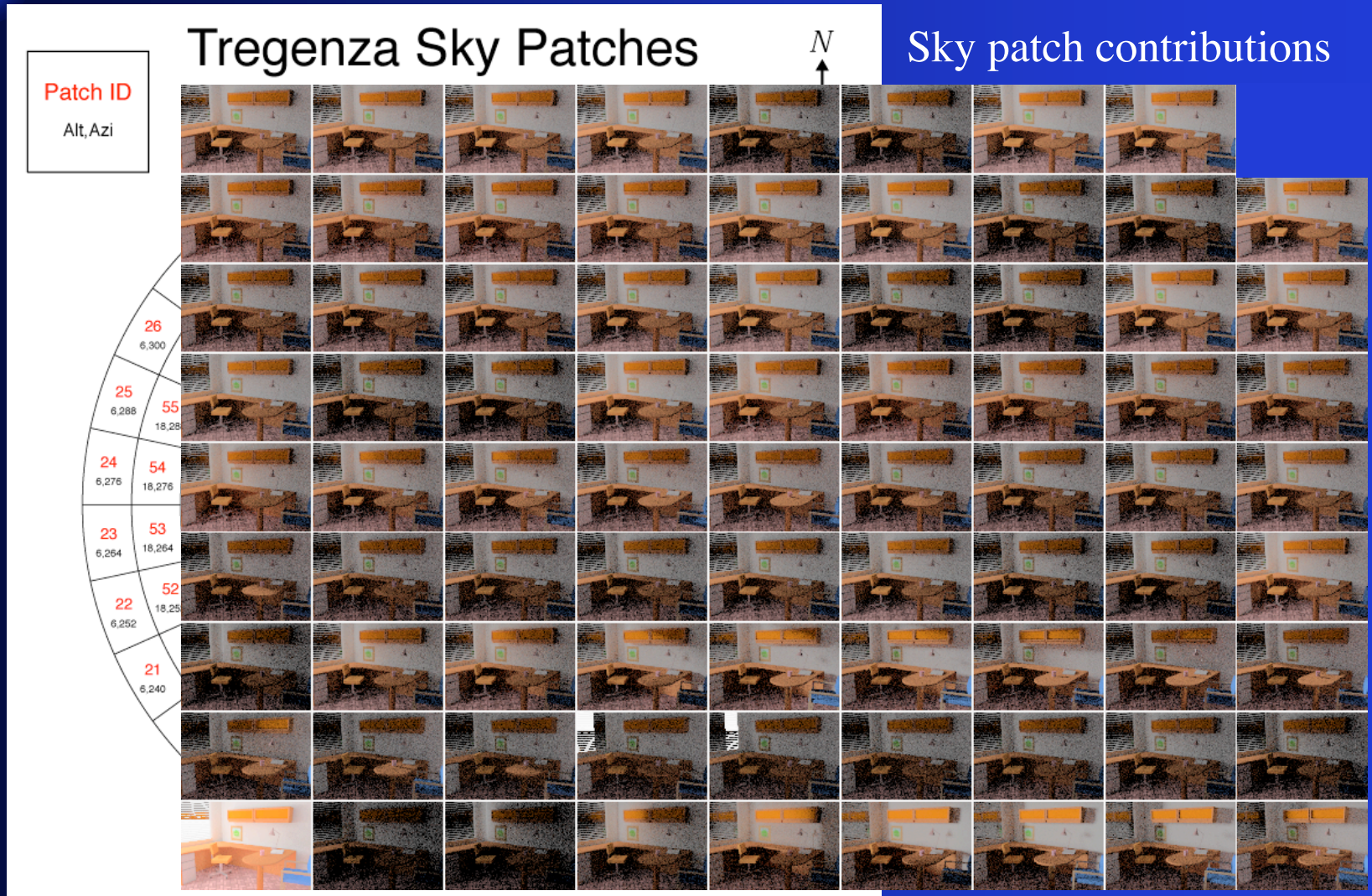
More Practical Example



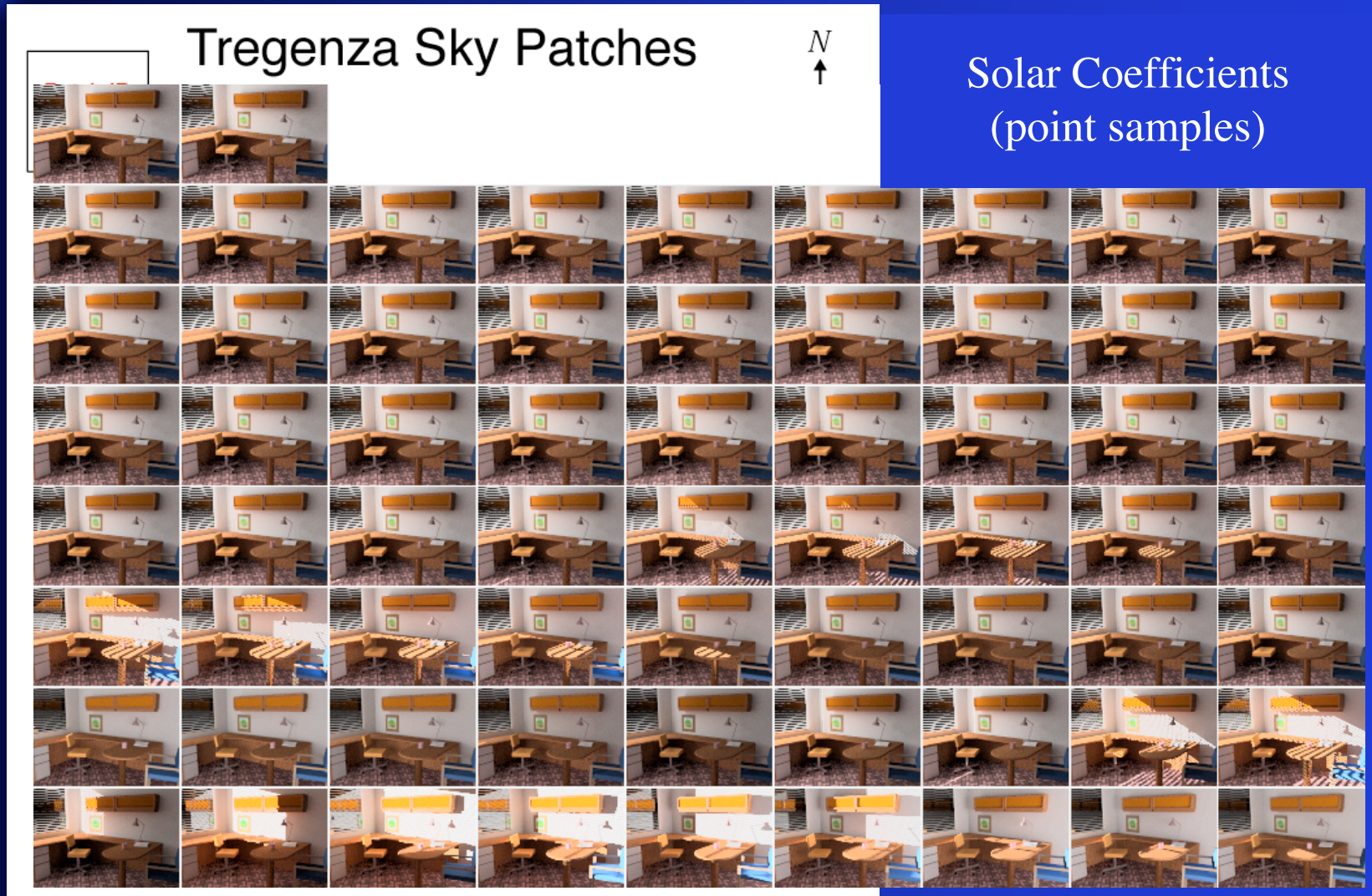
Tregenza's Daylight Coefficient Method:

Sky is divided into roughly equal-sized patches, and the relative contribution of each patch is calculated for a given geometry

More Practical Example



More Practical Example



Solar Coefficients
(point samples)

Summed Result from pcomb

Further Possibilities

- Have not tried computing BRDF's and BTDF's, but sure it can be done
- Might be possible to extend **mkillum** to handle curved, specular devices
- More details on how to use **rtcontrib** with a more extensive example in afternoon talk