# THE FUTURE OF *RADIANCE*

Greg Ward, Anyhere Software

Taoning Wang, Lawrence Berkeley National Laboratory

August 2022

# FIRST, A BIT OF HISTORY…

- Grew out of after-hours project at LBL to explore ray-tracing for lighting simulation
    - *Radiance* is named after the radiometric unit corresponding to a pixel
    - Indirect irradiance cache made it first practical physically-based ray-tracer ('87)
- Original source code distribution predates ANSI-C
    - Based on Kernighan & Ritchie standard, so no function prototypes
    - Most (but not all) code has been brought up to date with C conventions
- Development has been evolutionary -- no complete rewrites at any point
    - Many tools added over time, some retired (e.g., dot-matrix printer drivers)
- RGBE picture format was one of the earliest innovations
    - Enabled HDR imaging, including capture and image-based lighting

# NOW, THE STATUS QUO

- *Radiance* as it exists is validated, stable, supported on multiple platforms, and will be maintained by myself and others as long as there is interest in it
- It will continue to be useful in energy calculations, as a benchmark, and as an ingredient to other software
- DOE funding has been consistent but not constant over time, and may be waning
- There are alternative lighting tools (as always) and many are highly optimized
- The main strengths of *Radiance* are versatility and veracity (or validation, pick your ‘v’)
- Some of the code is new, but some code is 36 years old

# SYSTEM DESIGN AND CONSTRAINTS

- Single developer for multiple platforms, so focus on low-maintenance portability
  - “Least common denominator” approach -- relied mostly on standard C library
  - Minimize use of #ifdef’s and system dependencies
- Unix toolbox model:
  - Specialized tools communicating with well-defined file formats
  - File formats are standardized and (mostly) portable across architectures
  - Offers great flexibility -- adapts easily to new new applications
- Scene description language designed to be easy to read & write with printf() & scanf()
  - Less easy to read with eyeballs
- Later addition of “executive” tools such as **rad** to manage renderings, etc.

# THINGS THAT WORKED WELL FOR *RADIANCE*

- Sharing source code and interacting with a broad research & design community
  - Free/unrestricted source that can be understood and altered if necessary
  - E-mail questions usually got answered in a day or two
  - Contributions as well as critiques, suggestions & bug fixes came from many
  - Independent validation critical to acceptance and long-term value of simulation
- Unix toolbox model and standardized/portable file formats
  - Separate executables simplifies development and maintenance
  - Standard formats can be adapted and adopted by others (e.g., HDR pictures)
- Stable design allows long-term collaborations, incorporation into other software

# THINGS WE WISH WERE DIFFERENT

- Scene description format is clunky and not as general or extensible as it should be
    - E.g., RGB color model is “baked in” to material types, so no spectral specifications
    - MGF and similar description languages are potentially more powerful
- Single-developer model makes participation, progress slow
    - We need a way for coders to addand test new methods without breaking old ones
    - Unix toolbox is current model, but core simulation is too big for one module
- An updated system design (C++ library) would provide better modularity in core tools
    - Multiple rendering objects would provide more portable parallelization
    - Re-organization of rendering problem could enable GPU implementations

# TIMING

- As we said, *Radiance* is not going anywhere
  - Which is also part of the problem
- The question is: where would we like it to go?
- The time to plan for the future is now
- We have a community:
  - We can continue to help each other and benefit from collaboration
  - We can apply our collective expertise to new challenges

# SHARE YOUR PERSPECTIVE

- What are your current pain points?
  - As a researcher?
  - A practitioner?
  - A software developer?
- What’s your perfect ‘Radiance’?
  - E.g. how fast is it?
  - GPU-enabled?
  - Flexible and scriptable?
  - More accurate?
- What would be your most important new feature/addition?

# MOVING FORWARD

- How do we want to develop *Radiance*?
  - How do we want to manage development?
  - How do we deal with branches, variants?
- How do we want to fund *Radiance*?
  - Are there good self-sustaining models? Donations? Fees?
- How do we want to share *Radiance*?
  - What should the license look like?