Hi, my name is Thomas Ludwig. I’m with Glare Technologies, developers of Indigo Renderer.

We’re a small company of 3 fulltime employees, plus some friends and contractors helping out.

I’ve been with Glare for 10 years now, though the history of Indigo stretches back further than that as a hobby project of Nicholas Chapman.
I’ll start with a brief history of Indigo Renderer and our market context, and then go over some motivating examples for some of the design decisions.

Difficult indirect lighting, especially caustics, is not a focus of most rendering systems so I’ll go into detail about that, followed by user and developer perspectives for using bidirectional algorithms.
HISTORY AND CONTEXT

- Basis is Veach’s thesis [1], inspired by Maxwell Render
- Need for specialised product, “max quality” implementation
  - Make it accessible to non-CG specialists

Basis is of course Veach’s thesis, and Maxwell early pioneers in physically-based MC

Non-CG specialists e.g. architects, CAD designers, people with primary job in design, want good results easily

SketchUp and Revit users for archviz, C4D for productviz, Blender CAD
Indigo Renderer places great emphasis on image quality, and simplicity.

- Basis is Veach's thesis [1], inspired by Maxwell Render
- Need for specialised product, "max quality" implementation
  - Make it accessible to non-CG specialists
- Indigo Rendoror
  - Emphasis on quality and simplicity
  - (Volumetric) unidirectional and bidirectional path tracing
  - Optional Kelemen PSS-MLT [2] on top for most difficult scenes
  - Truly unbiased - 10k path depth, bidir on by default
  - Mainly archviz and productviz customers
MOTIVATION

- Archviz / productviz has different requirements and allowances
  - Can assume scene fits in memory, allows bidir methods and GPU
  - Demand highest final quality, quick previews
- Keep algorithms simple as possible
  - Need to exploit huge GPU resources
  - GPU unidir is already quite complex!

Biggest enabling assumption of viz: scenes can fit in memory

Relaxing this constraint allows powerful bidirectional methods

Pronounced advantage over unidir for rendering caustics

Fast early convergence big practical benefit of MLT, useful for previews
MOTIVATION

- Archviz / productviz has different requirements and allowances
  - Can assume scene fits in memory, allows bidir methods and GPU
  - Demand highest final quality, quick previews
- Keep algorithms simple as possible
  - Need to exploit huge GPU resources
  - GPU unidir is already quite complex!
- Interior renders much more efficient
  - Difficult to sample localised reflected light with eye paths
  - Light paths induce perfect distribution

For unidir to sample localised reflections, needs path guiding methods

When there is realistically modelled glass in front of emitter, you need bidir methods
Scenes as complex as this are not the norm, but same high accuracy engine rendering any archviz or productviz scene.
Many realistic scenes almost entirely illuminated by indirect light

Narrow IES lights particularly tricky for unidir
Green light path starts from the camera, hits the floor or wall, fails to make a direct lighting connection and strikes light through series of low probability scatters, producing firefly

Orange light paths start from emitter, reflect inside fixture, then strike floor or wall and connect to camera → perfect importance sampling
Some users willing to trade rendering speed / hw costs for simple setup, high quality final results

Nevertheless, all A/B comparisons in these slides rendered in 5 mins or less on desktop CPU

Bidir at best much more efficient, at worst not much worse (MIS), thus safest default without changing rendering modes
Saint Gobain needed predictive accuracy, measured spectral data for various commercially offered glass types

Indigo material library allows anyone to use extremely realistic glass
FROM A DEVELOPER’S POINT OF VIEW...

- Bidir is more difficult to implement and maintain
  - Unphysical hacks trickier
    - Section planes
    - Shadow catcher

Unphysical hacks commonly used for viz, e.g. section planes, invisible to cam objects

Section planes can’t simply clip off geometry, still need to emit from lights and reflect etc. More special cases for bidir

Shadow catcher planes are another special case, used for compositing
Similarly, more special cases for invisible to cam objects when using bidir

Refraction density different for light paths, interpolated normals need Veach’s “smoothing factor”

Fast N^2 implementation complex to implement and debug
AVOID COMPLEX ALGORITHMS

- Complex can mean:
  - Difficult to implement robustly, e.g. Veach-MLT vs PSS-MLT
  - Difficult to understand settings exposed to users, e.g. irradiance caching
  - Difficult to predict behaviour, e.g. flickering in animation
Vanilla bidir has overhead from many shadow rays, but there are ways to make it more efficient.

Combinatorial bidir, two-way path tracing by Simon Brown are examples.
Can’t ignore available GPU resources, factor ~10 brute force per GPU, GeForce RTX announced yesterday with dedicated RT silicon

Wavefront unidir PT already complex, bidir roughly doubles it (subpath tracing, MIS combining)

Would like to see more research on GPU out of core rendering, need practical method since complexity strictly additive
THANK YOU

References:

- [1] Veach thesis "Robust Monte Carlo Methods for Light Transport Simulation"
- [3] van Antwerpen thesis "Unbiased physically based rendering on the GPU"
- [4] Laine et al. "Megakernels Considered Harmful: Wavefront Path Tracing on GPUs"