Before we talk about path tracing: point clouds

- Sampling the real world:
  - Depth sensing cameras, e.g. Microsoft Kinect (2)
  - Laser scanners used in geospatial work

- Don’t get polygons: get points in space
  - But polygons can be fitted to point clouds . . .
  - . . . usable in games and real-time graphics?
  - Perhaps? [https://www.youtube.com/watch?v=Irf-HJ4fBls](https://www.youtube.com/watch?v=Irf-HJ4fBls)
Converging ends of a spectrum?

- We have frequently contrasted:
  1. Real-time, scan-line graphics
     - Z-buffers, Gouraud and Phong shading, ...
  2. Raytracing
     - “Photorealistic”, refraction, reflection, subsurface scattering, global illumination, depth of field (DoF) and other lens effects, etc.

- Modern hardware challenges this split!
GPGPUs return to graphics

- We’ve seen GPUs do scan line graphics
  - (That’s what they were explicitly built for.)

- OpenCL/CUDA has GPGPUs do maths
  - (General Purpose computing on GPUs)

- But raytracing involves lots of maths . . .
  - . . . can GPGPUs can help raytracing?

- How long does raytracing take?
Level of Detail (LoD) in scan-line graphics

- Scene ‘difficulty’ in scan-line graphics with hardware acceleration depends on:
  - Number of polygons and their size
  - Textures and fragment processing
  - Lighting calculation complexity

- Frame will be ready after some time
  - Determines frame-rate alongside the display’s refresh frequency
Level of Detail in raytracing?

- Raytracing difficulty depends on view
  - Mirror reflections of mirror reflections . . .

- But for a complex scene, we want to be able to ‘bail out’ in an elegant manner
  - But want some idea of scene appearance . . .
Runtime Path Tracing: Brigade

▶ Watch carefully the video at
https://www.youtube.com/watch?v=pXZ33YoKu9w

▶ Think about what you see within it:
  ▶ both in terms of good effects . . .
  ▶ . . . and also problematic artefacts
What did you notice?

- **Good:**
  - Mirror reflections / glossy surfaces
  - Indirect illumination
  - Depth of field

- **Bad:**
  - Speckled noise in the display
  - Frame rate
  - Still polygons (…is that actually bad?)
“May I have some path tracing?”

- The Brigade system is proprietary code
  - Valuable commercially while new technology
  - (and that’s for runtime Path Tracing)

- Look no further than Blender:
  - The ‘Cycles’ engine does Path Tracing
  - …planned to replace the internal ray-tracer
OK, so what *is* Path Tracing?

- Simplified: for a ‘sample’ of the scene:
  1. Set up primary ray as usual.
  2. Form new ray from random vector in hemisphere of hit-point normal.
  3. Recurse on that new ray (with depth+1).
  4. Add object emittance to recursion result, scaled by surface BRDF and reflectance.

- Repeat the above steps ‘many times’.
- Take the *average* colour of all samples.
Bidirectional Reflectance Distribution Function (BRDF)

- BRDF explains how light behaves at the surface of an opaque object.
- It is a function of light angle and view angle.
- Thus a generalisation of both diffuse and specular lighting.
- It is a four dimensional function (why four?).
BRDF vs. lighting we discussed

- Lambertian?
  - just ignore $\omega_r$
- Phong?
  - more than $\cos^n$:
  - get angular effects

Figure derived from Wikimedia Commons CC-SA media http://en.wikipedia.org/wiki/File:BRDF_Diagram.svg
Our path tracing was “simplified”

- Showed algorithm for ‘gathering rays’

- Can also use ‘shooting rays’ instead
  - Go from light source into the scene

- Better: ‘bidirectional path tracing’
  - Both gathering and shooting; connect paths
We said “repeat many times”

- OK, so approximately how many times?

- For gathering rays we want thousands of samples per surface point.
- For shooting rays we want orders of magnitude more than that.
Samples: “use average colour”

- Raytracers average supersampled colours
  - But scenes look OK without supersampling—
  - lighting models fairly completely explained (maybe poorly) the illumination at the hit-point.

- Path Tracing process is more of a statistical process.
  - This is the cause of the speckle noise.
  - ‘Eventually’ the image will converge.
Beyond BRDF

- Want refraction?
- Need BTDF: the Bidirectional Transmittance Distribution Function

Derived from public domain Wikipedia image
Limitations

▶ Sub-surface scattering:
  ▶ Need the Bidirectional surface scattering reflectance distribution function (BSSRDF)

▶ Pathological caustics:
  ▶ Can use Metropolis Light Transport

▶ Colour effects: basic Path Tracing still won’t simulate a prism correctly