Simple Ray Tracer with the Apache Beam Go SDK

Robert Burke (@lostluck) Beam Summit 2021
Learning Goals

- How do Ray Tracers work
- How to write one with the Beam Go SDK
- How to use SplittableDoFns with it
- Debugging Beam Go
- Executing on a Distributed Runner
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects
What is a Ray Tracer?

- Simulates the physics of Light to generate images
- Does it backwards
- Can achieve subtle and complex effects
Additional rays are cast
Further Additional rays are cast
Further Additional rays are cast
● Read in the scene and its configuration options
● Set up the camera
● For each pixel:
  ○ Cast sampling rays from the camera to the scene
  ○ Find the object in the scene the ray intersects with
  ○ Depending on the properties of the object
    ■ Cast additional sampling rays to determine the color of the object
      ● These can be called “bounces”
    ■ Stop when we hit the bounce limit
  ○ Accumulate the contribution from all sampling rays
  ○ Set the pixel color
● Save the image
https://github.com/lostluck/experimental/
Dividing Work
Splittable DoFn

- Create a Restriction for an element
- Split a Restriction for a given element appropriately.
- Create Trackers for a restriction.
- Process the element with respect to the given restriction tracker.
Splittable DoFn

Element: The image being produced.

Restriction: Offset Ranges

- Enumerate each Sample from 1 to Width*Height*Samples.
- Decompose from numbers back to individual pixel coordinates.
- Easy to Split, built into Beam
type Ray struct {
    Xp, Yp, Zp float64 // Position
    Xv, Yv, Zv float64 // Vector
    Rc, Gc, Bc float64 // Color
    Xpx, Ypx int32    // Pixel
    Bounce, ID int16 // SampleID
}
The Problem
The Problem

\[ 8M_{px} \times 4096_{rays/px} \times 88_{b/y} = 3 \text{ Tb} \]
Debugging Beam Go Pipelines
Local Debugging

- Unit test your code
- Counters
- Local portable runners and LOOPBACK mode
- Profile your code
Counters
Local Runners and LOOPBACK Mode
Distributed Execution

1. Main Program starts up and constructs the pipeline object.
2. Sends the worker artifact etc to the runner.
3. As needed, Runner starts SDK Worker and Runner Worker containers, which fetch the worker artifact.
4. Runner assigns workers bundles to execute until termination
Distributed
Local Execution

1. Main Program starts up and constructs the pipeline object.
2. Sends the worker artifact etc to the runner.
3. As needed, Runner starts SDK Worker and Runner Worker containers, which fetch the worker artifact.
4. Runner assigns workers bundles to execute until termination.
1. Main Program starts up and constructs the pipeline object.
2. Runner tells Main program to start a LOOPBACK server, to create SDK Workers
3. Sends the worker artifact etc to the runner.
4. Runner spins up workers in the main program process via Loopback.
5. Runner assigns workers bundles to execute until termination
Starting a job in LOOPBACK mode

Loopback mode enabled
Profiling
Add calls to `pprof.StartCPUProfile(f)` and `defer pprof.StopCPUProfile()` to your main()

$ <execute job locally, in LOOPBACK with profiling>
$ go install github.com/google/pprof
$ sudo apt-get install graphviz
$ pprof --http=: <binary name> <profile file name>
Distributed Runners
$ go run . --use_beam=true --word=GOOGLE \ 
--samples=1024 --bounces=5 \ 
--runner=dataflow \ 
--project=$PROJECT \ 
--staging_location=$STAGING_GCS \ 
--region=us-central1 \ 
--job_name=rebo-gbrt3 \ 
--environment_config=$SDK_CONTAINER
Job run on Google Cloud Dataflow
Job run on Google Cloud Dataflow
Learning Goals

- How do Ray Tracers work
- How to write one with the Beam Go SDK
- How to use SpliitableDoFns with it
- Debugging Beam Go
- Executing on a Distributed Runner
Fabien Sanglard and Andrew Kensler
https://fabiensanglard.net/postcard_pathtracer/

The Beam Summit organizers

Viewers like you.
\[ L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + \int_{\Omega} f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t) (\omega_i \cdot n) \, d\omega_i \]