LRT overview

Bill Mark
CS395T
Sept 2, 2003

Irt

- Physically based renderer
- Designed to be extended
  - This is good and bad…
- Well documented
- Good **algorithmic** optimizations
- **Not** micro-optimized
- Designed for static scenes ☹
  - We'll fix that later

Assignment #1

- Assignment:
  - Instrument lrt to gather coherence data
  - Summarize this data in histograms
- Purpose:
  - Get familiar with lrt and tools
  - Understand coherence properties of lrt
- Logistics:
  - OK to work with partner
  - Only high-level discussions with other groups
  - Due at *start* of class, 9 days from now
  - Turn in a written writeup

Four ways to think about Irt

- Geometry of the scene
  - Where are rays?
  - In what order do we trace the rays?
- C++ class structure of code
- File/directory organization of code
- Run-time “trace” through code
  - Single-step in debugger

Geometry of the scene

- One eye ray at a time
  - With differential information
- At surface hit, may recursively trace additional rays
- (Figure on whiteboard)

Key physics-related classes

- Geometric/physical data
  - **Ray::** = information about a ray
  - **Radiance::** (L) = power / (area * solid angle)
    - **Spectrum::** = three such values
  - **Primitive::** = collection of surface geometry
    - **GeometricPrimitive::** = single geometric primitive (e.g. sphere)
  - **Light::** = photon emitter

- Basic physics computations
  - **SurfaceIntegrator::**
  - **VolumeIntegrator::**

Scene:: = main bag of stuff

- Data
  - Geometry in scene (type=Primitives)
  - Lights in scene (type=Light)
  - Camera (type = Camera)
  - Surface integrator to use (SurfaceIntegrator)
  - Volume integrator to use (VolumeIntegator)
  - Etc.
- Methods
  - **Render**

Main rendering loop – **Scene::Render** method

Setup;

Loop over samples {

Create camera ray for this sample

Evaluate ray's radiance (i.e. trace the eye ray)

Add sample contribution to image

}

Trace a ray – **Scene::L** method

- Determine radiance arriving along a ray
- Does all necessary work, including recursive tracing of rays, etc.
- When ray hits a surface, **Scene::L** invokes the appropriate **SurfaceIntegrator**

One surface integrator **Whitted::**

- This is a plugin
  - Lives in lrt.src/integrators/whitted.cc
- Code:

  Find intersection of ray with surface
  (Scene::Intersect → Primitive::Intersect)

  Compute radiance from reflection of direct illumination
  (loop over lights;
  weight each by BRDF/BSDF if not shadowed)

  Trace specular reflection ray

  Trace specular refraction ray

Lots of details omitted

- Fast ray/object intersection
  - **Accelerator** : optimized data structure and methods to allow quick **Intersect** queries.