Ray Tracer System Design & Irt Overview

cs348b
Matt Pharr
Overview

• Design of lrt
  • Main interfaces, classes
  • Design trade-offs
• General issues in rendering system architecture/design
• Foundation for ideas in remaining lectures
Key Design Features

• Plug-in architecture
  • Run-time object loading
  • Don’t need to recompile entire system to add functionality
  • Strict enforcement of OO interfaces

• Carefully-chosen abstractions
  • Based on fundamental physical quantities
Basic Rendering Process

• Parse scene description & create representation
• Simulate light transport, render image
• Apply imaging pipeline, write out result
Rendering Interface

- Well-defined interface between user and renderer
- Two classic approaches
  - Describe *what* to render (RenderMan)
  - Describe *how* to render it (OpenGL)
- *What* is more elegant (if you can afford it)
  - Curved surfaces basic surface description (not triangles)
  - Physically-based light models
  - Materials
Rendering Interface

- Hierarchical graphics state is very convenient
- Less so if exporting scene from modeling app
- Begin/end state stack model
- RI flattens it for use by the renderer
- Overall task is to create appropriate objects
Runtime Instance Creation

• Instance creation based on name/ParamSet
  • RI knows little about specifics of available plugins
• ParamSet encapsulates name/value pairs
  • Type declaration
  • Value setting/getting
Basic Geometric Classes

- Point, vector, normal
  - Important to differentiate between them
- Ray
- Transform
- Operator overloading to make it easy to transcribe equations:
  - \( v = p1 - p2 \);
  - \( \text{ray}(t) \)
  - \( p1 = \text{transform}(p2) \)
Other Basic Utility Classes

• Spectrum
• Memory allocation
  • Cache-aligned allocation
  • Memory pools
  • Reference counting
• Float2Int
• Random numbers
• Statistics
Key Abstract Classes

- Instances created by rendering interface
  - Primitives
    - Shapes
    - Materials
  - Accelerator
  - Lights
  - Camera
  - Sampler
  - Integrators
Main Rendering Loop

- Scene object holds all the objects from RI
- Scene::Render()

```cpp
while (more samples) {
    get next sample
    generate camera ray
    compute radiance along ray
    update image
}
apply imaging pipeline
```
Sampler

- Drives image sampling
  - Jittered, low-discrepancy, dart throwing, ...
- Key task: good anti-aliasing
  - More samples: better image
  - Sample positioning very important
- Sample encapsulates sample position
  - image, time, lens, integration...
- Rendering continues as long as it makes more samples
Camera

• Encapsulates viewing/imaging properties
  • Turns samples into rays
  • Projective, orthographic, spherical, ...
• May simulate depth of field
Integrator

- Process kicks off with rays from camera
- Computes radiance along given rays
  - Many different levels of accuracy/realism
  - Appel: camera rays + shadow rays
  - Whitted: camera rays, shadow rays, specular reflection rays
- Two stage process
  - Geometric: find closest intersection
  - Radiometric: compute reflected light
Primitive

- Represents basic geometry & its material
- Given ray, return Intersection, if any
- May also refine, like Shapes
- GeometricPrimitive
  - Shape, Transform to place in scene
  - Material
Shape

• Quadrics, triangle mesh, subdivision surface
• Refine() key for complex shapes
• DifferentialGeometry represents ray intersection
  • Point
  • Normal
  • (u,v)
  • Tangents
  • ...

Matt Pharr, Spring 2003
Accelerators

• Implemented behind Primitive interface
• “Meta-hierarchies”
• Grid, kd-tree
  • Implementation made more tricky by refinement option, however
Materials

- Spatially-varying surface reflection characteristics
- Texture describes variation
- Task is to return BSDF at intersection point
  - ("Bidirectional scattering distribution function")
BSDF

- Reflection at a single point
  - Reflected light from integrating incident light times reflection function
- Local coordinate system simplifies implementation
Texture

- Modulate spatially-varying material properties
  - Texture map
  - Procedural texture
  - Constant value
- Texture tree representation
- Anti-aliasing and filtering a key responsibility
Light

• Emission of visible energy into the scene
• Classic graphics lights
  • Point, distant, spot, ...
• Area lights
• VisibilityTester closure to defer shadow ray tracing
Imaging Pipeline

• Compensate for display limitations
• Floating-point to integer conversion
• Quantization, dithering, gamma correction
• Tone reproduction