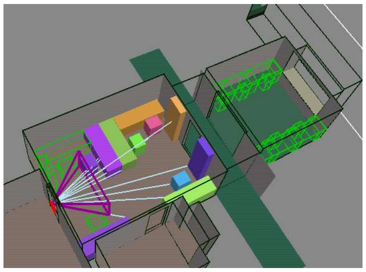
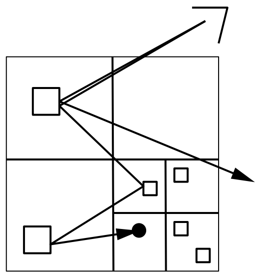
Lecture 21: Tuesday, 23 November 1999

Accelerating Ray Casting & Ray Tracing





Announcements:

- Office hours today, after class
- No class Thursday (enjoy the break!)
- Next Tuesday, November 30^{th}:
  - Visible surface determination II
  - HKN evaluations (final 15 minutes)

Observations about Ray Casting

Performance
- Most effort (> 90%) on ray/object intersections
- So ... worth investigating acceleration techniques

Coherence
- Spatial coherence
  - Nearby rays have correlated object intersection
  - Nearby ray trees have similar topologies
  - Rays often cast in "pencils" (e.g., view frustum)
  - Nearby rays produce similar (correlated) radiances
  - Rays not uniformly distributed in ray space
  - Memory footprint (working set) changes slowly
- Temporal coherence
  - Ray query implies nearby recent/imminent queries
  - Set of cast rays varies smoothly, predictably
  - Scene objects, lights move smoothly, predictably

Desiderata for Acceleration Methods

Single frustum, or multiple (animation, simulation)?
- If multiple, is viewer path known a priori?

Static or dynamic scene?
- If dynamic, are object motions known a priori?

Preprocessing allowed?
- Batch time, and storage space available?
- Otherwise, on-line with little or no latency

Space-time, quality-time tradeoffs?
- How much space available for data structure?
- Can correctness be sacrificed for performance?

Ray query distribution?
- Ray Casting (regular pencil of eye rays; coherent)
- Ray Tracing (eventually divergent after multiple bounce
- Radiosity (for form factors; locally coherent)
- Visibility (e.g., inter-character in simulation)
- Application-dependent (e.g., walking observer)
- Invariant, or highly variable, over time ?

Basic Approaches to Acceleration

0. Parallelize
   - Issues of prediction, latency, etc.
1. Make each ray/object intersection faster
   - Build spatial data structures for queries
2. Process rays as correlated, generalized groups
   - Determine behavior of groups of rays
3. Make fewer ray/object intersection queries
   - Interpolate sparse samples
     - With or without correctness guarantees

1. Accelerating Individual Intersections

Bounding hierarchy: **organize space by objects**
- Issues & Implications:
  - Choice of volume (box, sphere, slabs, etc.)
    - Test vs. bounding volume, or object
  - How to construct bounding hierarchy
    - Model semantics; component proximity; volume/are
  - Static / Dynamic / Adaptive ?
    - Object insertion/deletion/motion; query dist.
- To show: bounding hierarchy, clustering up

1. Accelerating Individual Intersections

Spatial subdivisions; **organize objects by space**
- Issues & Implications:
  - Choice of partition method
    - Branching factor, regularity (BSP, octree, k-d)
  - Construction algorithm
    - Incremental (bottom up); Global (top down)
- To show: BSP construction/query; octree/k-d tree

2. Process Rays as Groups

Beam-tracing **organize rays by space**
- Cast pencils of rays with similar behavior
- To show: 2D beam propagation

Issues & Implications:
- Construct pencils by object (Heckbert & Hanrahan '84)
- Restriction to polyhedral objects
- Construct pencils by screen region (Alex et al. '98)
- Reflection, refraction induce non-linearity

Also: trace cone, radiance samples

2. Process Rays as Groups

Ray subdivision: **organize queries by parameter**

5-DOF ray space subdivision (Arvo & Kirk '87)
- To show:
  - Construction (2D example)
  - Query (by ray origin, direction)
- Tradeoff: space- or object-based subdivision
  - Relative advantages / disadvantages ?

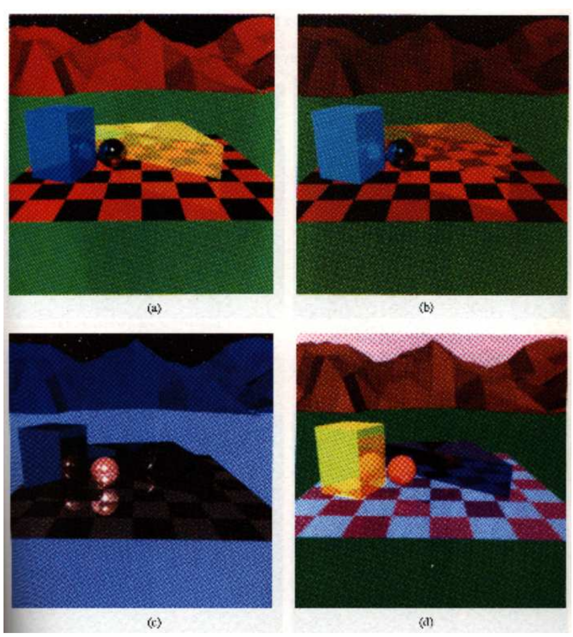
2. Process Rays as Groups

Object-driven: **organize queries by objects**
Example: shaft culling (Haines & Wallace '91)
For systems that ray cast for form factors
Visibility data structure per interaction
Note: triage, adaptive construction

3. Reusing ray/object intersections

Parametrized Ray Tracing (Séquin & Smyrl '89)
Radiance at root of ray tree is function of:
material properties, light sources, trig
So: *fix* ray tree, alter materials/lights !
Rapid re-evaluation of pixel values
Optimizations: common trees, subexpressions
Careful: can't use usual truncation criteria!

3. Example (from Séquin & Smyrl '89)



(b) light to blue; (c) mountain/plain/shiny; (d) all colors changed.
For more complex scene (see paper):
Base time: 4900 seconds, 8Mb
Parametrized time: 5640 seconds, 28Mb
Redisplay time: 30 to 81 seconds

4. Reusing ray/object intersections

Radiance Interpolants (Bala et al., '97 – '99)
Subdivide ray-space *per-object* (only 4D)
Prevent interpolation across samples when:
Topology of ray-tree varies, or
Interpolation incurs error $> \epsilon$
To show: ray-space, interpolation predicate

Characterizing acceleration schemes

Bottom-line: how much does application's ...
- Run-time decrease ?
- Render-time decrease ?
- Memory footprint decrease ?
- I/O traffic decrease ?

Memory / speed tradeoff
- Best: controllable footprint

Time / quality tradeoff
- Best: controllable quality

Fixed/adaptive: sensitive to query distribution?
- Best: move frequent queries up in decision tree
- Careful: may yield unacceptably high variance

Generalizations

Dynamic scene
- Must repopulate objects into subdivision
  - Treat index as cached/memoized ray queries
  - Must invalidate entries involving moved objects

Multiprocessor architectures
- Lazy computation (give up correctness)
- Fill ray cache with predictive queries
- Eject samples with LRU/geometric criterion