Lecture 20: Thursday, 13 November 1997

Today:
Accelerating Ray Casting / Ray Tracing
Visibility Computations
Hidden Surface Elimination
Visible Surface Identification

[NEW BOARD]
Review of Approaches:

1) Make each Ray/Object intersection faster

2) Trace different entity (beam, cone, radiance sample)

[NEW BOARD]
Approaches:

3) Make fewer ray/object intersection queries

[global visibility data structure:
determine fate of many rays at once
cell/portal subdivision, frustum advance
-- defer until discussion of cell/portal subdiv]

classify many rays at once
frustum subdivision for ray casting
works well for casting -- not tracing

spawn fewer samples; interpolate for reconstruction
fixed coarse subdivision
adaptive subdivision
how to know when to subdivide/terminate ?
look at variance of corners, but...
prove something about interior

[NEW BOARD]
4) Generalizations:

dynamic lights/material properties
store full color expression at each pixel
use common subtree elimination for compactness
careful: can't apply usual truncation criteria!

dynamic scene geometry
repopulate moving objects to subdivision
treat it as cache; clear valid bit
must also clear entries involving this object

multiprocessor architecture:
fill ray cache with predictive queries
eject with LRU or geometric criterion

[NEW BOARD]
Visibility Computations: Related problem

given a polyhedral scene, and a dynamic viewpoint
efficiently compute the set of visible polygons

[SHOW input: background, + two polys which together block a fourth,
and a fifth poly totally occluded behind one of the first two]

Three classes of algorithms: analytic, ordering, superset
analytic: produces exact set of visible fragments
[SHOW fragments -- convex or not, in color]

ordering: visible object is last one drawn at each pixel
[SHOW output on example input: 1 2 3 4]

superset: produces upper bound on visible set
-- assumes a z buffer for per-pixel visibility
[SHOW four of five are output, in any order]
Two kinds of superset algorithm:
Additive: assume nothing visible; explore world
Useful (for example) in architectural interiors
Subtractive: assume everything visible; identify occlusion
Useful (for example) in urban exteriors

Another dimension of variation:
Object space: operates in continuous coordinates
(can often conclude visibility info about view region)
Screen space: operates to screen resolution
(typically, z-buffer based, requires viewpoint)
Hybrid: some combination of both

holy grail:
visibility algorithm for generic scenes
indoor scenes with evident structure
outdoor scenes with large occluders

dense scenes with no obvious structure:
complex CAD model (engine compartment)
complex organic scene (forest)
statistical scene elements (fog, etc.)
AND, which allows predictive (region) visibility queries

historical reasons: show in order analytic, ordering, superset

Analytic:
Pure object space:
classical computational geometry problem:
O(n^2) lower bound -- show two combs input
one way: sweep-line alg, emit visible fragments

Pure screen space: ray casting, one ray per pixel
subject of last lecture

Warnock algorithm: interesting hybrid
attach all polygons to root viewport
subdivide polygons, and clip away from children
when reach region with
zero polygons ? done.
one polygon ? render poly to whole region.
one pixel ?
compute z's, render closest only
or... compute analytic solution in pixel

Ordering:
show BSP tree construction
choose a poly; split on it, assoc. w/node
if other poly is split, partition it

show tree traversal
traverse far side, split plane, near side

problems ?
BSP tree can be O(n^2) size; all nodes traversed!
also, can take O(n^3) time to construct !
what if polygons form a cycle: A > B > C > A ...

Superset:
indoor environments:
cell/portal subdivision -- show construction
recursive frustum propagation (point superset vis)
stabbing portal sequences (region superset vis)

populate with ''detail objects'', show tests

works well in densely occluded environments
with easily-identified structural occluders

problem: occlusions might be compound/hard to detect

outdoor environments:
first, show sparse occlusion, why cell-portal breaks

so: consider a large occluder, vs. dynamic eyepoint
casts large occluding shadow volume
objects in shadow volume need not be rendered
or, pairwise, consider large, small objects
when eye is in supporting tangent region
large object occludes small objects

in both cases, question is: how to detect efficiently?

subdivide space with k-d tree
associate good occluders with each cell
when in cell, use these occluders
on-line: form shadow volume of eye, occluders
subject octree to occlusion cull, recursively

temporal coherence: common tangents of occluder, occludee
identify cells contained in region; track viewpoint

future directions: visibility certificates, priority queue