1996-12-01 Bo Nordin (bosse@cb.uu.se)

Lecture notes, computer graphics (F15) (Foley 12.1-12.4, Foley 13)

Have discussed simple 2D / 3D primitives
From now one: less detail goal is no longer to be able to write pgms but to understand!

Visual Realism (12)

What is realism? (photorealism, visual cues)
Is it really necessary?

Pipeline: 1) create models (chap's9, 10), 2) viewing spec(chap6) & lighting
3) VSD (13) 4) shading (14)
Rendering - creating images from models (rendering pipeline)

Realism important in simulation, animation, films etc.
But why create virtual realities?? (Cheaper than creating real. Research. Simulation)

Fundamental problems:
the world is complex
possible if the goal is to understand the 3D relationship. But still projection 3D -> 2D.

Rendering line drawings (showing 3D depth, 12.3)
several views (easy if simple, but plate 24...)
perspective (smaller object farther away)
depth cueing
depth clipping
texture & colour
visible-line determination
animation

Rendering shaded images (12.4)
visible-surface determination (today)
illumination, shading, shadows, transparency, reflection (next time)

Visible-Surface Determination, VSD (Intro)

Problem: given a set of 3D objects we want to determine which surfaces/lines are visible, either from the COP (persp.) or along the direction of projection (par.)

Simple concept, but significant computing power needed - numerous algorithms exist.
Many special-purpose hardware architectures.

Image precision (at the resolution of the display device):

```
for (each pixel) {
 determine the object closest  to viewer pierced by the projector
  draw the pixel
}  /* effort prop to n*p where n is # of objects and p is # of pixels */
```

precision of the display / aliasing / redo it all if zoomed
originally for raster (256x256 then)

Object precision (at the precision of the object):

```
for ( each object ) {
  determine unobstructed parts of objects
```

draw those parts
} /* prop to n*n but considerably more complex */
precision with which the objects are defined
must be followed by the actual display (mapping to screen)
only this last step must be redone for a different resolution
originally for vector (4096x4096)

First some general remarks on how to organise the work efficiently

Coherence

Take advantage of local similarities (in the environment or in its projection)
properties vary smoothly, it is the very discontinuities we exploit to produce effects.
object coherence - completely separate objects need not be compared in detail (in z)
face coherence - surface properties vary smoothly
incremental algorithms from pixel to pixel
sometimes faces do not interpenetrate
edge coherence - visibility changes only when crosses behind a visible edge or penetrates a visible face
(implied) edges can be computed from 2 points of intersections between planes
scan-line coherence - set of visible objects spans in one scan line differ little from the next
area coherence - adjacent pixels often covered by the same face
depth coherence - adjacent parts close in depth.
frame coherence (animation) - successive frames differ little

Perspective transformation

VSD must be done in 3D space before throwing away the z as we've done so far
Given two points, does either point obscure the other (are they on the same projector?)
Depth comparison *after* normalising transformation
Parallel projections easy
Problem with perspective transformation: $x_1/z_1 == x_2/z_2$ and $y_1/z_1 == y_2/z_2$.
Can be avoided by first transforming the object to the 3D screen-coord. system (pg. 445):

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1+z_{min}) & -z_{min}/(1+z_{min}) \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

transforms the norm persp. truncated-pyramid to a parallelepiped -1 <= x,y <= 1, -1 <=z <=0 clip in hom coordinates

This transformation preserves relative depth, straight lines, and does the perspective foreshortening. Normally incorporated in N_{per} & clip in hom. coord.
If we do this we remove the problems of fig 13.1 (will assume in the following)

Extents and bounding volumes

screen *extents* are used not only to smart up clipping, but also to avoid unnecessary comparisons between objects. If M above used we can check simply the (x,y)-boxes of proj. extent testing similar to trivial reject testing in clipping (*bounding boxes*)
bounding volumes used to surround the objects themselves rather than their projections
extents can also be used to test whether objects overlap one 1 dim (z, for instance).
the most complicated part is finding the extent!!
can also be used to test whether a projector intersects an object

Backface culling

Assume all polygons are defined such that their normals point out from the polygon

In eye coordinates, a back-facing polygon may be identified by a non-negative dot product that its surface normal forms with the vector from the centre of projection.
If an scene consists of only one convex polyhedron (or the boundingbox test has eliminated the rest) backface culling is all we need to do.

Ax + By + Cz + D = 0 (we need 3 points or 2 vectors on a face to compute A..D)
direction of projection is (0,0,-1) -> backfacing if the surface normal has a neg z coordinate (.i.e. C < 0), if == 0 the polygon is viewed on the edge - i.e. invisible surfaces may have visible edges

Complex objects may have self-hidden surfaces
Halves the # of polygons for each pixel in ip methods

Spational partitioning
(or spatial subdivision) large problem -> several smaller ones
divide the projection plane with a 2D grid and determine in which grid spaces a proj lies
adaptive partitioning can be used if objects are unequally distributed
hierarchical models (object at a level can serve as extent for its children)

z-Buffer algorithm
(or depth buffer) IMAGE precision
Need a frame buffer and a z-buffer (equally big)
init z-buffer = 0 (back clipping plane)and framebuffer to bgr colour
during scan conversion (3.5), if a point is no farther away than the stored pixel the new pixel replaces the old one (alg, program 13.1, page 451)
no pre-sorting or object-object comparisons needed
an exhaustive search
use depth coherence: normally we solve $z_0 = (-D - Ax - By) / C$, $z_1 = z_0 - A * \Delta x/C$
and a similar technique may be used to determine first z on the next line

simple alg. if memory is expensive, we can scan convert in strips but in that case we must perform multiple passes through the objects. Aliasing (because of image precision)

z-buffer can be kept and used to merge in new objects
mask-off

Scan-line algorithms
IMAGE precision - one scan line at a time
extension of the polygon scan-line algorithm
1/ create an edge table (ET) of all non-horizontal edges
entries sorted into buckets based on each edge's smaller y coordinate
and within buckets ordered by increasing x coordinate of their lower endpoint.
each entry: 1/ the x coord corresponding to the smaller y coordinate
2/ the y coord of the other end
3/ the x increment Δx (inverse slope)
4/ The polygon id
2/ create a polygon table 1/ ID
2/ The coeff's of the plane equation
3/ shading/colour info
4/ an in-out flag, initialized to FALSE
3/ We also need an active edge table (AET) - kept in order of increasing x

pg 455-456 (simplified if polygons do not penetrate each other)
to use it for penetrating polygons break up the polygons and introduce *false edges*

one can also use a single-line frame buffer and z-buffer (simple is it)

Visible-surface ray casting (tracing).
Prototypical image precision algorithm

List-priority algorithms

LPA determine a visibility ordering for objects, and ensure that objects are rendered in that order. Simple: if no objects overlap in z, simply render them in z order.
If objects overlap or penetrate we must split objects to make a linear ordering possible
LPA's combine image precision and object precision:
depth comparisons and object splitting are done in op
scan conversion is done in ip

Depth-sort algorithm: paint the polygons into the frame buffer
1/ sort all polygons based on the smallest (farthest) z coordinate
2/ resolve ambiguities (z overlaps), split polygons if necessary
3/ scan convert each polygon in ascending order of smallest z (back to front)
Painter's algorithm (Correct for objects that exist in a plane of constant z, 21/2D)
(skip step 2 and pray...) (cf. explicit priority)
What about the step 2? Let P be the current at the far end, test against Q
Test a polygon against all others, five tests (if a test succeeds they do not overlap) :
0/ do the polygons' x extent not overlap
1/ do the polygons' x extent not overlap
2/ do the polygons' y extent not overlap
3/ is P entirely on the opposite side of Q's plane from the viewpoint (draw and explain)
4/ is Q entirely on the same side of P's plane as the viewpoint
5/ do the projections onto the (x,y) plane not overlap
if all tests fail we repeat 3 and 4 reversed. (and move Q to the end of the list if OK)

Binary space-partitioning trees:
efficient for applications where objects are the same and the viewpoint varies.
behind root<- Root -> in front of root
can be used for an arbitrary viewpoint!!

Area-subdivision algorithms
divide-and-conquer spatial partitioning
Warnock (recursive subdivision): subdivides each area into 4 equal squares.
At each stage a polygon has one of four relationships to the area:
1/ surrounding polygons completely contain the area
2/ intersecting polygons intersect the area (as contained after clipping!)
3/ contained polygons are completely inside the area
4/ disjoint polygons are completely outside the area
In four cases we do not need to subdivide the polygon further:
1/ all polygons outside (disjoint)
2/ only one intersecting or only one contained polygon
3/ single surrounding polygon
4/ more than one polygon but a surrounding polygon in front.
determining this is done by computing the z value of all relevant polygons at the four corners

if none if these is true -> subdivide (down to pixel resolution,
or even further for antialiasing)

Comparisons

Sorting is the heart of VSD. Important to avoid unnecessary sorting (use coherence!)
depth sorting sorts on zxy in that order
scan line sort in yxz
Warnock sorts on x/y and then z (xy)z
z-buffer is called an (xyz) algorithm

Few polygonal faces: depth sort best, z-buffer worst
Many polygonal faces: z-buffer best, depth-sort worst

BSP <- new views of a static environment simple
z-buffer <- adding an objects to the same view simple, simple alg, - memory-eater