Visible-Surface Determination
(Hidden Surface Removal)

Computationaly expensive

Two basic types: image-precision and object-precision

For n objects and p pixels

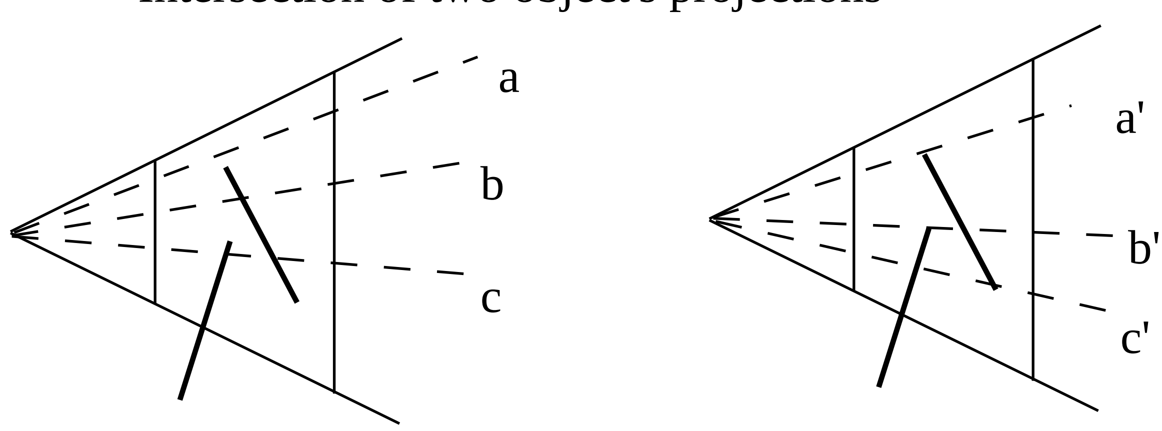
Image-precision
- For each pixel, determine which object is visable
- Requires np operations

Object-precision
- For each object, determine which part(s) hidden by other parts or objects
- Requires n^2 operations

Which is more efficient?
- n < < p
- complexity of the "operations"
- what happens if change the size of the display?

Increasing efficiency

Basic computations:
- Intersection of a projector and an object
- Intersection of two object's projections

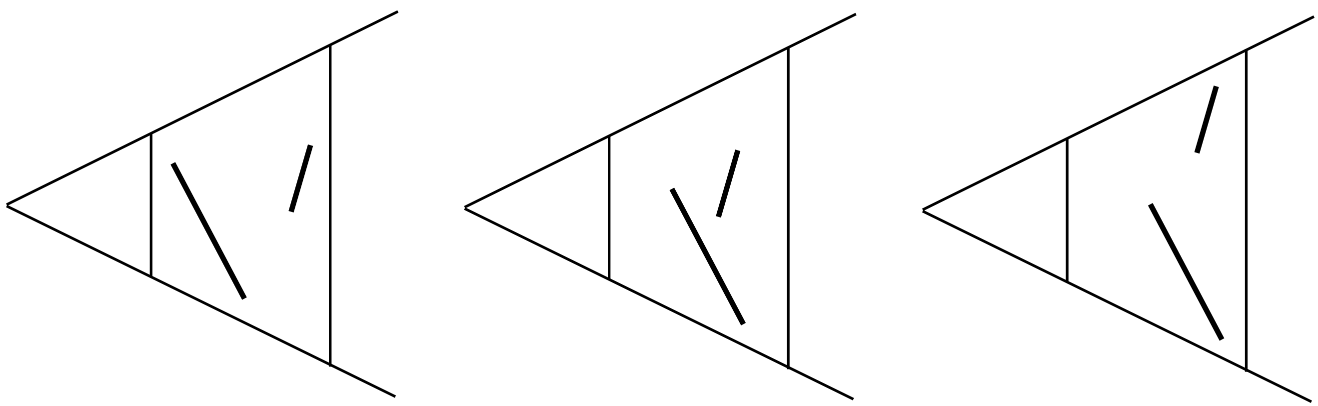


Use coherence to simplify the computations

Coherence - the degree to which parts of an environment or it's projections exhibit local similarities

Object coherence

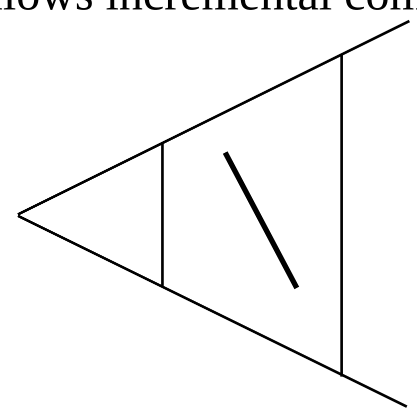
all of object A may have the same relation to all of object B



Face coherence

surface properties vary smoothly across a face

allows incremental computations

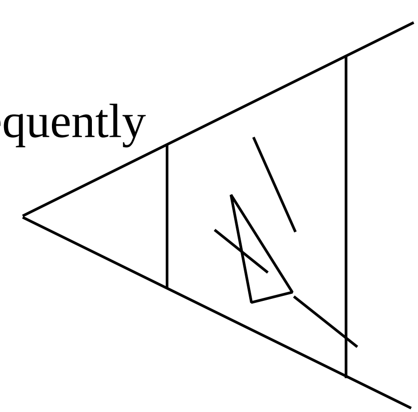


ie, depth varies smoothly across the face

Edge coherence

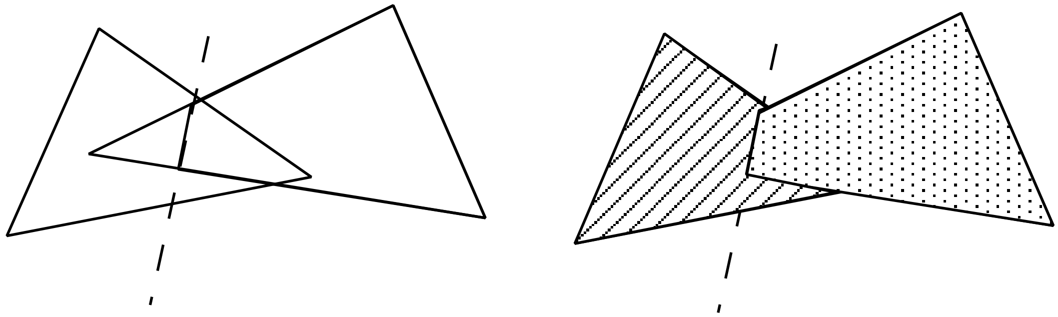
an edge changes visibility infrequently

where?



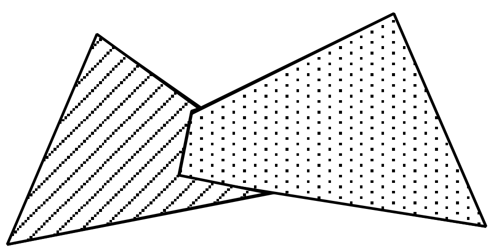
Implied Edge Coherence

when two planar faces intersect, their line of intersection (their implied edge) can be determined from two points on the intersection



Scan-line coherence

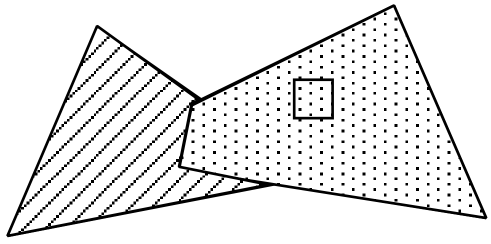
The set of visible spans of objects differs little from one scan-line to the next



For how many scan lines will there be big changes? (assume no accidental alignment)

Area coherence

A group of adjacent pixels is often covered by the same object

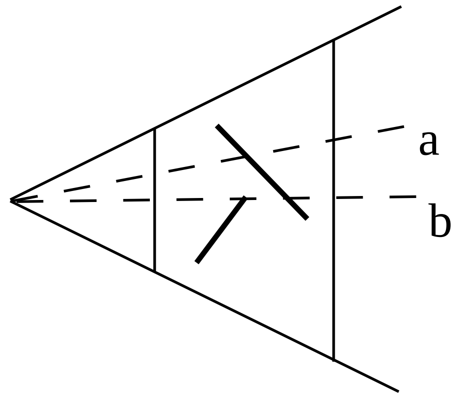


When not true?

Depth coherence

neighboring points on a single surface have similar depth

adjacent screen points corresponding to different surfaces usually have different depths



Frame coherence

Pictures of same environment at t and t + 1 will be similar

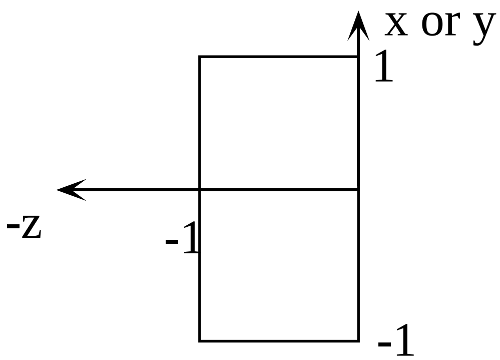
Animation - in general most of scene remains the same

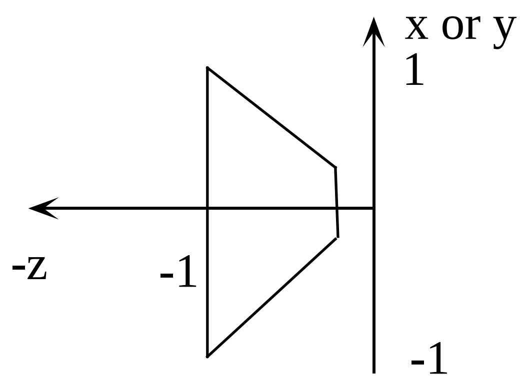
The perspective transformation

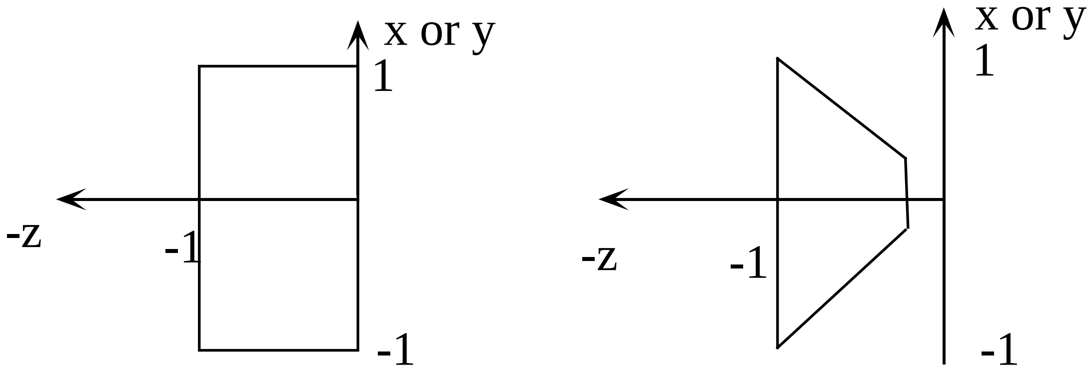
Can you do visible surface determination in 2D screen coordinates?

Can you do visible surface determination in 3D world coordinates before applying the projection transformation?

If do it after the normalizing transformation has been applied (Normalizing transformation transforms the view volume into the canonical view volume)







For parallel projection:

How tell if points on the same projector?

$x_1 = x_2$ and $y_1 = y_2$

For points on same projector, how tell which is visible?

compare z values

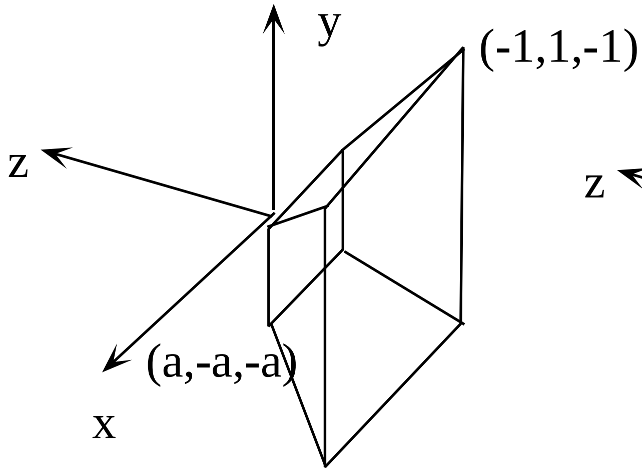
For perspective projection:

How tell if points on the same projector?

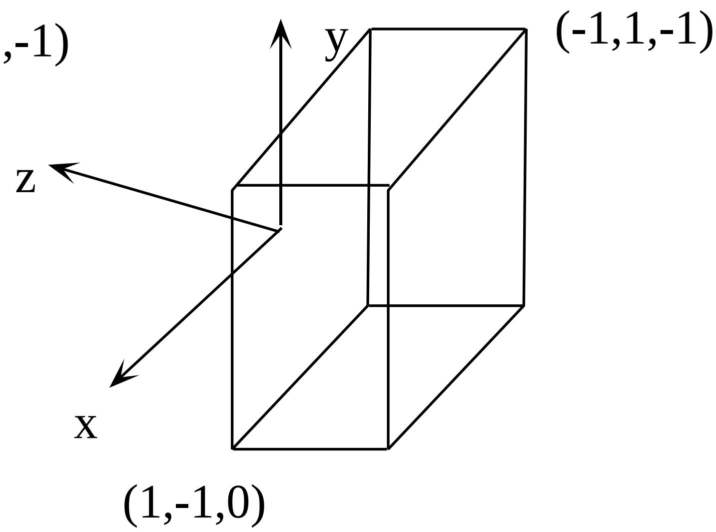
$x_1/z_1 = x_2/z_2$ and $y_2/z_2 = y_1/z_1$

Thus need to do four divisions - expensive

Avoid this by first transforming in 3D screen-coordinate system



Canonical Perspective
View Volume



View Volume in
3D screen coordinates

To transform between these two spaces:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1+z_{min}) & -z_{min}/(1+z_{min}) \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad 0 > z_{min} > -1$$

Why is it easier to do hidden surface removal in the 3D screen coordinate system?

When to clip?

Apply N_{per} to get canonical view volume,

then clip

then apply M and do hidden surface removal

versus

Compose N_{per} and M and apply

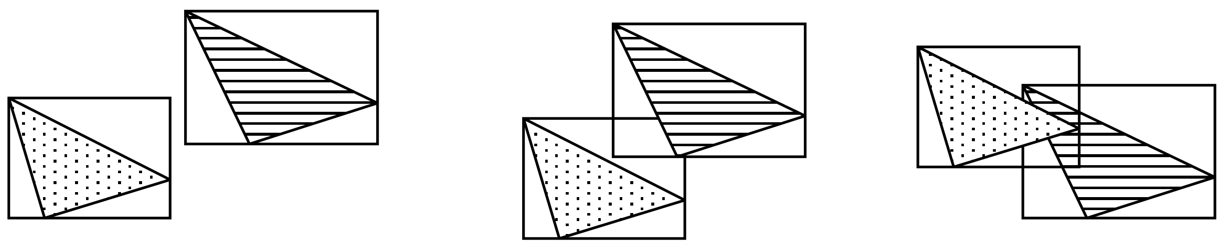
then clip

Is M necessary for parallel projections?

Extents and Bounding Volumes

Simplify visible surface determination by using extents

Compute extents (bounding boxes) of objects
in 2D screen coordinates



If extents don't overlap - no object occlusion
If extents do overlap - object occlusion?

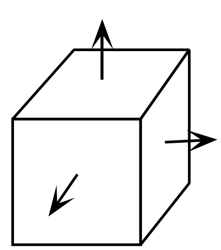
Also simplifies computing intersection of projector and object

If projector doesn't intersect extent - no intersection
If projector does intersect extent - intersection with object?

Back-face culling

For solid polyhedron objects

Define surface normal for each face of polyhedron
(points out from polyhedron)



In eye coordinates,
face not visible if dot product of surface normal
and projector to any point on surface is nonnegative

Assume:

- n_f = outwards surface normal of a face
- q = any point on the face
- e = eye position (center of projection)

Then direction of projector is:

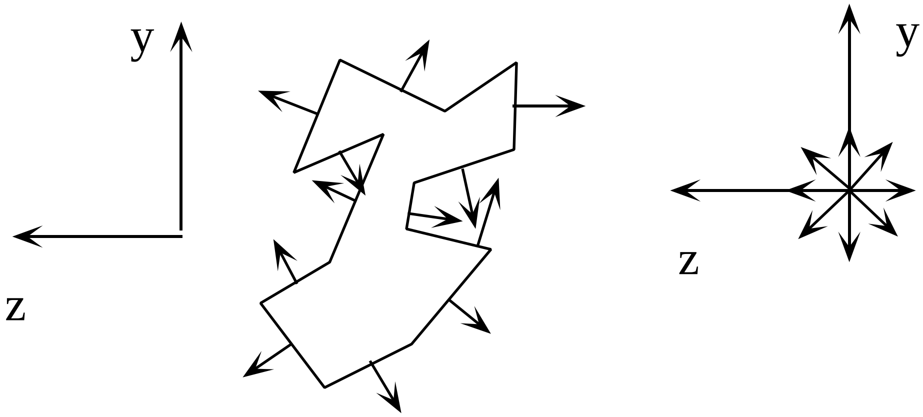
- v = vector from e to q

Face is not a back face if:

- angle between vectors v and n_f is <= $^{1}/2$
- dot product of v and n_f >= 0

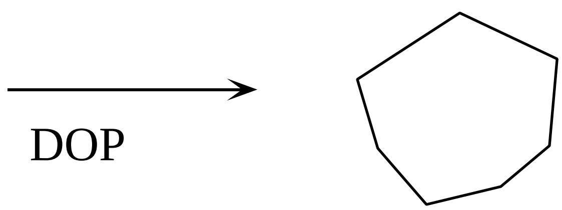
In 3D screen coordinates

If surface normal has negative z coordinate, then not visable

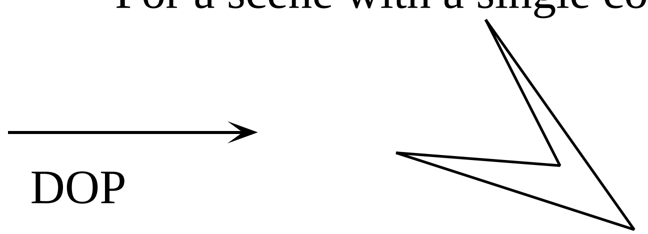


Is this all we have to do for hidden surface determination?

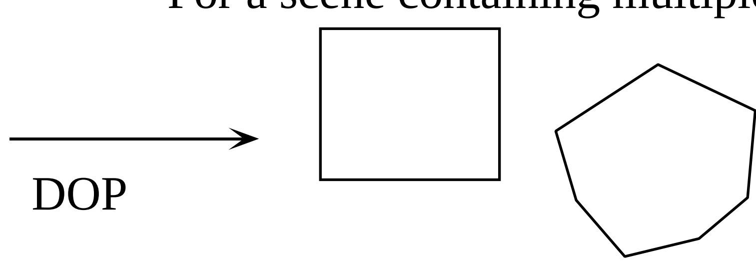
For a scene with a single convex polyhedron?



For a scene with a single concave polyhedron?



For a scene containing multiple polyhedrons?

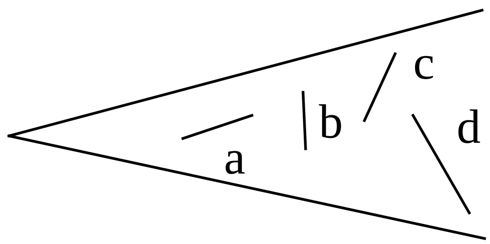


Painter's algorithm

Sort surfaces by their depth

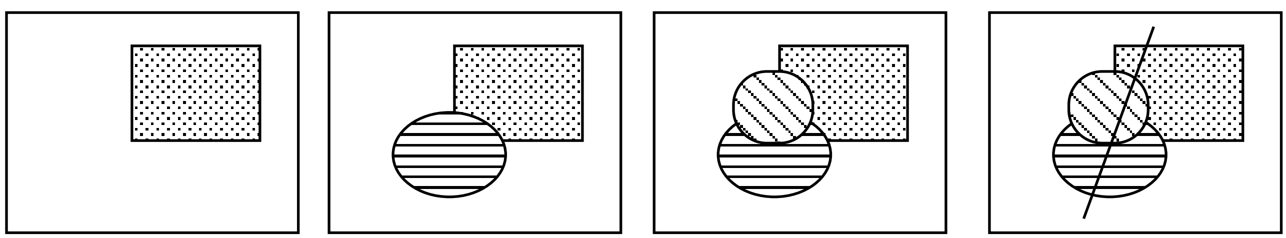
Render in order of furthest to closest

Close surfaces overdraw the further surfaces



Rendering order:

d, c, b, a



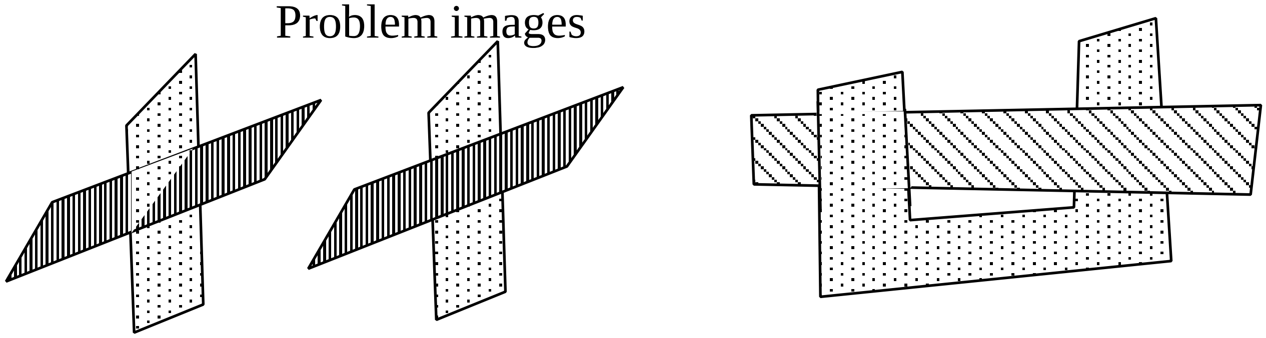
When will this work?

Let min_a be the minimum depth of surface a

Let max_a be the maximum depth of surface a

What has to be true for the painter's algorithm to always be correct?

Problem images

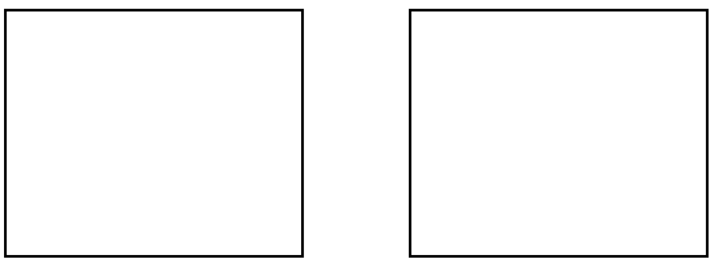


The z-Buffer Algorithm

Image-precision algorithm

Catmull (1974)

Requires additional z-buffer of same width and height in image pixels with depth corresponding to required depth precision of scene



Frame Buffer   z-Buffer

Initialize all values in frame buffer to background color

Initialize all values in z-buffer to z value of back clipping plane

For each pixel in each polygon,
    If z is closer than value of that pixel in z-buffer,
        Write z into pixel's location in z-buffer
        Write polygon pixel value into frame buffer

Very simple algorithm to implement
Not space efficient
Not time efficient

How improve space efficiency?
    Use a scan-line sized z-buffer instead of image sized z-buffer

    What is the additional cost of this?

Use depth coherence to reduce computational cost

Since polygons are planar,
if know the depth of one point on a scan line,
then simple to find depth of next point if on same plane

for plane defined by:

$$z = (-D - Ax - By) / C$$

if at (x,y),

$$z = z_1$$

then at (x+1,y)

$$z = z_1 - (A/C)(\Delta x)$$

so just one subtraction
(A/C) is a constant
$\Delta x = 1$

Can do same thing for finding first z value on next scan line

To get cutaway views:

when do z-buffer algorithm, don't update frame buffer and
z-buffer if depth is in front of cut plane

Do you need to recompute the z-buffer for this?

How to add a new object to a scene projection?

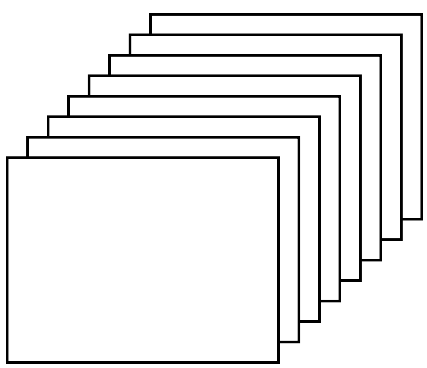
If saved z-buffer,
just apply z-buffer algorithm to the new object using
old z-buffer and frame buffer

How to add and remove a new object?

If want new object in scene, and then be able to remove it without
rescanning entire scene

Use old z-buffer, but don't update it when scanning new object

Use overlay frame buffer



Example: move little box of given depth around the scene to get idea of depth of objects