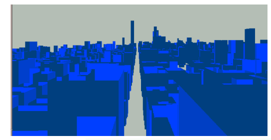
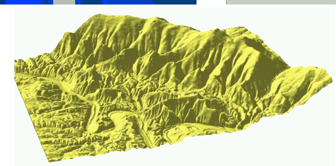
Rendering Surfaces

- ☆ *Visible surface determination*
- ☆ *Illumination and shading*
  - Shade individual images
  - Ambient light
  - Complex light sources.
- ☆ *Material properties*
- ☆ *Modeling curved surfaces*
- ☆ *Improved illumination models*
- ☆ *Texture and bump mappings*
- ☆ *Shadows*
- ☆ *Transparency and reflections*
- ☆ *Improved camera models*

Slide 1

Visible Surface Determination





Σ Given a 3D scene (set of polygons)

p Viewpoint

F View frustum

Compute the portion of Σ visible from p within F.

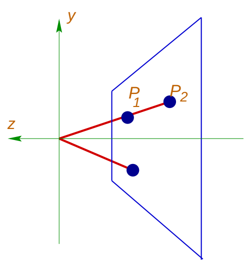
Perspective view: p is at finite distance.

Orthographic view: p is at infinity.
Only viewing direction is relevant.

Slide 2

Occlusion

- ☆ Viewing direction is along the $(-z)$-axis.
- ☆ P_1 and P_2: Two points in 3D.
- ☆ P_1 *occludes* P_2 if
  - P_1 and P_2 lie on the same projector, and
  - P_1 is closer to the center of projection.



- ☆ If P_1 and P_2 are *not* on the same projector, they do not occlude each other.

Slide 3

Canonical View Volumes

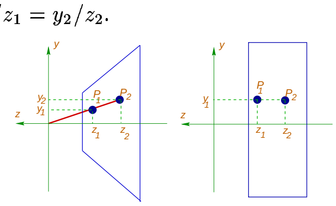
View volume affects the projectors, which define *occlusion*.

Perspective view volume:

$$P_1 = (x_1, y_1, z_1), P_2 = (x_2, y_2, z_2)$$

P_1, P_2 on the same projector if

- ☆ $x_1/z_1 = x_2/z_2$,
- ☆ $y_1/z_1 = y_2/z_2$.



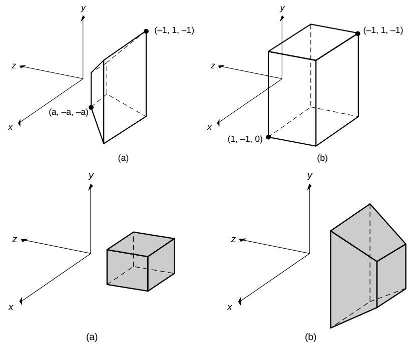
Orthographic: P_1, P_2 on the same projector if

- ☆ $x_1 = x_2$,
- ☆ $y_1 = y_2$.
- ☆ No divisions!

Slide 4

Canonical View Volumes

Transform perspective view volume to orthographic view volume.



Slide 5

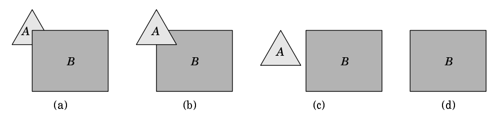
General Approaches

Σ Set of objects.

Π Set of pixels.

Object-space approach:

for (each object $\sigma \in \Sigma$) {
  Find the visible part σ' of σ
  Draw the pixels corresponding to σ'
    in color of σ;
  }

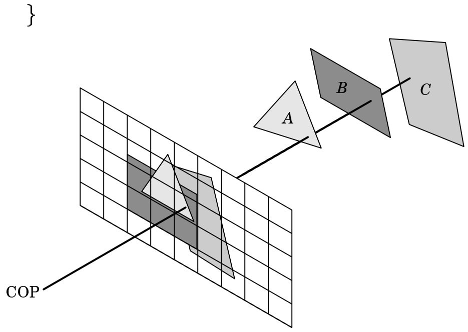


Slide 6

General Approaches

Image-space approach:

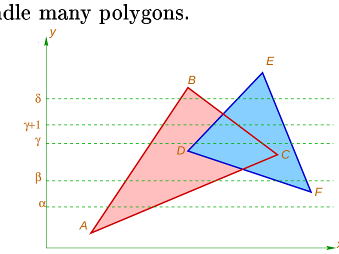
for (each pixel $\pi \in \Pi$) {
  Find the closest object σ pierced by
    the projector through π;
  Draw π in color of σ;
  }



Slide 7

Scan-Line Algorithm

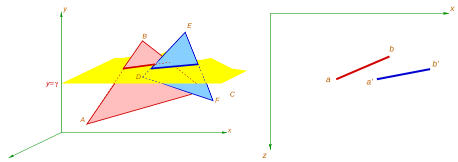
Generalize the polygon scan-conversion algorithm to handle many polygons.



- ☆ Scan from bottom to top.
- ☆ For each scan line, visit the pixels from left to right.
- ☆ For each pixel π determine the polygon σ closest to the viewpoint.
- ☆ Draw π with the color of σ.

Slide 8

Edge Coherence



- ☆ Scan line corresponds to a horizontal plane h (parallel to the xz-plane) in 3D.
- ☆ Intersection of h and a polygon P is a line segment ab.
- ☆ Endpoints of ab are intersection points of the scan line and the edges of P.
- ☆ Let $q_1, q_2, \ldots$ be the intersection points of projections of polygon edges and the scan line.
- ☆ Same polygon is visible between q_i and q_{i+1}.

Slide 9

Scan-Line Algorithm

Data structures:

- ☆ *Edge table*: Same as in polygon scan conversion.
- ☆ *Polygon table*: Each entry stores
  - Coefficients of the plane equation.
  - Shading and color information.
- ☆ *Active edge lists*

Scan line	Entries
α	AB, AC
β	AB, AC, FD, FE
$\gamma, \gamma+1$	AB, DE, CB, FE
δ	AB, CB, DE,FE

Slide 10

Z Buffer Algorithm

- ☆ Simple to implement in hardware.
- ☆ Requires two buffers.

Frame buffer $F[x, y]$: color of pixel (x, y).

Z **buffer** $Z[x, y]$: z-coordinate of the object visible at pixel (x, y).

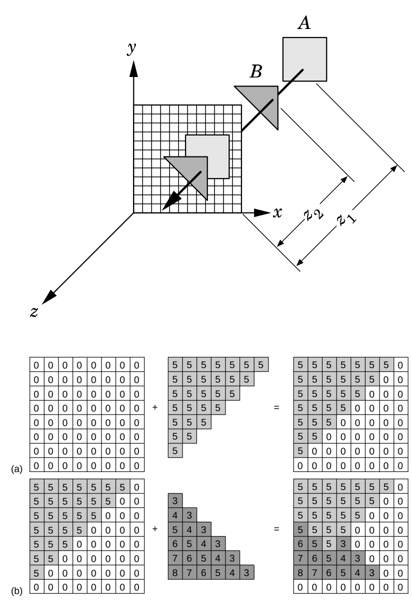
Initially, each element in the Z buffer stores the z-value of the *far clipping plane* of the view volume.

Σ: Set of polygons.

```
for σ ∈ Σ {
    for each pixel p = (x, y) in projection of σ {
        z_p = z-value of σ at p;
        if z_p > Z[x, y] {
            Z[x, y] = z_p;
            writePixel (x, y, σ's color at p);
        }
} }
```

Slide 11

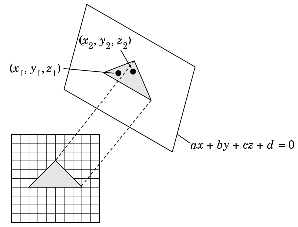
Z Buffer Algorithm



Slide 12

Z Buffer Algorithm

Use polygon scan conversion algorithm to process the pixels of σ.



Plane equation of σ: $Ax + By + Cz + D = 0$.

$$z = \frac{-D - Ax - By}{C}$$

$$z(x+1, y) = \frac{-D - A(x+1) - By}{C} = z(x, y) - \frac{A}{C}$$

Between scan lines, y increments by 1, so

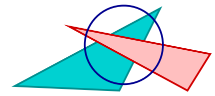
$$z(x, y+1) = z(x, y) - \frac{B}{C}$$

Slide 13

Z Buffer Algorithm

Disadvantages

- ☆ Uses only the front-most polygon to set pixel colors.



- ☆ Cannot handle transparent objects.
- ☆ *Aliasing problems*: Different polygons may share the same pixel.
- ☆ *Roundoff errors*
  - Perspective view to orthogonal view transformation reduces z precision.
  - Polygons with different depths may have the same z-value.
  - Static objects may *swap occlusion* as the camera moves.

Slide 14

Z Buffer in OpenGL

glClear(GL_DEPTH_BUFFER_BIT);

glEnable(GL_DEPTH_TEST);

glDepthFunc(GLenum *fn*);

GL_NEVER	GL_ALWAYS
GL_LESS	GL_LEQUAL
GL_GREATER	GL_GEQUAL
GL_EQUAL	GL_NEQUAL

- z-value: Distance between the object and the viewpoint.
- If z-value of the new *fragment* satisfies *fn*, its z-value is written in the depth buffer.
- Default is GL_LESS.

glDepthMask(*mask*);

- GL_TRUE: z-buffer in read/write mode.
- GL_FALSE: z-buffer in read only mode.

Slide 15

Blending

- ☆ Without blending a pixel is overwritten in the frame buffer (objects are *opaque*).
- ☆ Blending allows to combine the existing color of a pixel with that of incoming fragment.
- ☆ Blending allows to display transparent/translucent objects.
- ☆ A (Alpha) in RGBA mode specifies blending.
  glColor4f (R, G, B, A)
- ☆ Smaller values of A denote higher transparency.

Example: **glColor4f** (1.0, 0.0, 0.0, 0.2)
$A = 0$: transparent; $A = 1$: opaque
☆ Red glass with 80% transparency.
☆ Can have Multiple transparent objects.

Slide 16

Blending

Source: Incoming fragment.

Value: (R_s, G_s, B_s, A_s)

Blending factor: (S_r, S_g, S_b, S_a)

Destination: Stored pixel.

Value: (R_d, G_d, B_d, A_d)

Blending factor: (D_r, D_g, D_b, D_a)

New value of pixel:

$(R_sS_r + R_dD_r,\ G_sS_g + G_dD_g$
$B_sS_b + B_dD_b,\ A_sS_a + A_dD_a)$

`glEnable`(GL_BLEND)

`glBlendFunc`(*sfactor*, *dfactor*)

Slide 17

Blending Factors

Constant	S/D	Blend Factors
GL_ZERO	S/D	$(0,0,0,0)$
GL_ONE	S/D	$(1,1,1,1)$
GL_DST_COLOR	S	(R_d,G_d,B_d,A_d)
GL_SRC_COLOR	D	(R_s,G_s,B_s,A_s)
GL_ONE_MINUS_DST_COLOR	S	$1-(R_d,G_d,B_d,A_d)$
GL_ONE_MINUS_SRC_COLOR	D	$1-(R_s,G_s,B_s,A_s)$
GL_SRC_ALPHA	S/D	(A_s,A_s,A_s,A_s)
GL_ONE_MINUS_SRC_ALPHA	S/D	$1-(A_s,A_s,A_s,A_s)$
GL_DST_ALPHA	S/D	(A_d,A_d,A_d,A_d)
GL_ONE_MINUS_DST_ALPHA	S/D	$1-(A_d,A_d,A_d,A_d)$
GL_SRC_ALPHA_SATURATE	S	$(f,f,f,1)$
		$f=\min\{A_s,1-A_d\}$

Slide 18

Blending: Examples

Example 1: Combining two images with equal blending factor.

- ☆ Draw the first image with
  sfactor = GL_ONE
  dfactor = GL_ZERO
- ☆ Draw the second image with
  sfactor = GL_SRC_ALPHA
  dfactor = GL_ONE_MINUS_SRC_ALPHA
  $A_s = 0.5$

Example 2: Image through a photographic filter that blocks 20% red light, 60% green light, and 28% blue light.

- ☆ Set destination color

$$(R_d, G_d, B_d, A_d) = (0.8, 0.4, 0.72, 1.0).$$

- ☆ Draw the image with
  sfactor = GL_DST_COLOR
  dfactor = GL_ZERO.

Slide 19

Handling Transparent Objects

Σ: Scene (set of polygons) consisting of

- ☆ Opaque objects
- ☆ Transparent objects

☆ Draw all the opaque objects

- Depth buffer in the normal write mode

☆ `glDepthMask`(GL_FALSE); (Read only mode)
`glEnable`(GL_BLEND); (Turn on blending)

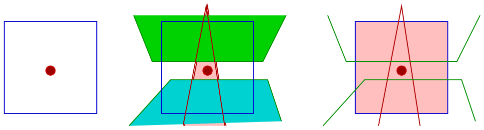
☆ Draw translucent objects with blending.

- Translucent objects behind an opaque objects do not have any effect.
- Translucent objects in front of all opaque objects do not change the z-value.
- Colors are blended.

Slide 20

Blending and Antialiasing

- ☆ Pixel π is not a point.
- ☆ z-value is not the same over the entire pixel.
- ☆ Compute z-values at the center of π.



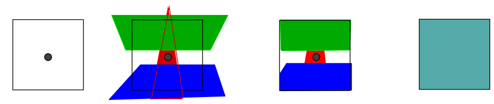
- ☆ Compute all polygons $\sigma_1, \ldots \sigma_k$ visible at π.
  C_i: color of σ_i.
  Assume σ_i's sorted by their depth value.
- ☆ For each polygon, compute the area A_i of σ_i visible within π.
- ☆ Blend the colors accordingly.

$$C(\pi) = \sum_{i=1}^{k} C_i A_i$$

Slide 21

Blending and Antialiasing

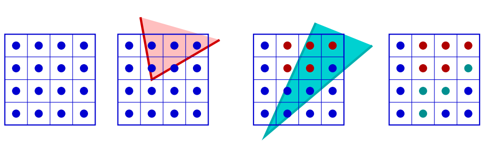
$\Pi = \pi$
`for` $(i = 1,\ i \le k,\ i{+}{+})$ {
  $\Pi_{\text{front}} = \Pi \cap \sigma_i$
  $C = C + \frac{Area(\Pi_{\text{front}})}{Area(\Pi)} \cdot C(\sigma_i)$
  $\Pi = \Pi \setminus \Pi_{\text{front}}$
}



Slide 22

Antialiasing and Blending

- ☆ Computing Π_{front} and Π are expensive!
- ☆ Use a *mask* (M) (super sampling) for each pixel.
  (*Typical masks size:* 4×8).
- ☆ Run the z-buffer algorithm on all the subpixels.
- ☆ Blend the colors of the subpixels to compute the color of a pixel.



Slide 23

Buffers

- ☆ Frame (color) buffer
- ☆ Depth buffer
- ☆ Accumulation buffer
- ☆ Stencil buffer

Slide 24

Accumulation Buffer

- ☆ Analogous to multiple exposures.
- ☆ Sequence of images is generated.
- ☆ Images are accumulated into the *accumulation buffer*.
- ☆ After accumulation the result is copied back to the frame buffer for viewing.
- ☆ Accumulation buffer has higher precision.

Antialiasing

```
for (i = 1, i ≤ μ, i++) {
    Offset the image by i-th subpixel position;
    Draw the shifted image;
    Accumulate the color values into the A-buffer;
}
Divide the color values of each pixel by μ;
Draw the normalized image;
```

Slide 25

Accumulation Buffer in OpenGL

```
glClearAccum(R, G, B, A);
glClear(GL_ACCUM_BUFFER_BIT);
glAccum(op, val);
```

Buff: Value of the frame buffer.

GL_LOAD	Acc = Buff ∗ *val*
GL_ACCUM	Acc = Acc + Buff ∗ *val*
GL_RETURN	Buff = Acc ∗ *val*
GL_ADD	Acc = Acc + *val*
GL_MULT	Acc = Acc ∗ *val*

Slide 26

Accumulation Buffer

Antialiasing

```
for (i = 1, i ≤ ACSIZE, i++) {
    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);
    accPerspective (···);
    displayobjects ();
    glAccum(GL_ACCUM, 1.0/ACSIZE);
    }
glAccum(GL_RETURN, 1.0);
```

Motion Blur

- ☆ Offset the image by the motion of objects.
- ☆ Different objects can move at different speed.
- ☆ Entire scene can be made dimmer by
  `glAccum(GL_MULT, decay)`

Slide 27

Accumulation Buffer

Depth of field

- ☆ Objects lying only on a particular plane are in focus in a picture.
- ☆ All objects are in focus in a scene drawn by OpenGL.
- ☆ Draw the scene repeatedly with slightly different values of `glFrustum` and accumulate the results.
  - Viewpoint is slightly different
  - Objects on a particular plane remain focused

Soft shadows

- ☆ Draw the shadows from each light source separately
- ☆ Accumulate the results.

Slide 28

Stencil Buffer

- ☆ Restricts drawing to a certain portion of the image plane.
  E.g.: Drawing a scene through an odd-shaped window
- ☆ **glStencilFunc** (func, ref, mask)
  - *func*: Comparison function to decide whether a pixel should be drawn.

GL_NEVER	GL_ALWAYS
GL_LESS	GL_LEQUAL
GL_GREATER	GL_GEQUAL
GL_EQUAL	GL_NEQUAL

  - *ref*: The value stored in stencil buffer is compared with *ref* using *func*.
  - *mask*: Both *ref* and stencil-buffer are bitwise ANDed with *mask*.
- ☆ **glStencilOp**(fail, zfail, zpass)
  - Specifies how the data in the stencil buffer is updated when a fragment passes or fails the test.

Slide 29

Stencil Buffer

- Possible options:

GL_KEEP	GL_ZERO
GL_REPLACE	GL_INCR
GL_DECR	GL_INVERT

- *fail*: Function used when the stencil test fails.
- *zfail*: Function used when the stencil test passes but the depth test fails.
- *zpass*: Function used when both the stencil and depth tests pass.

Slide 30