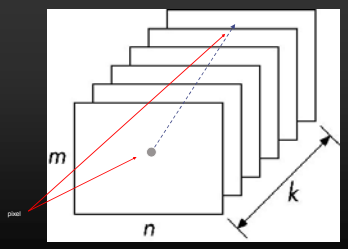
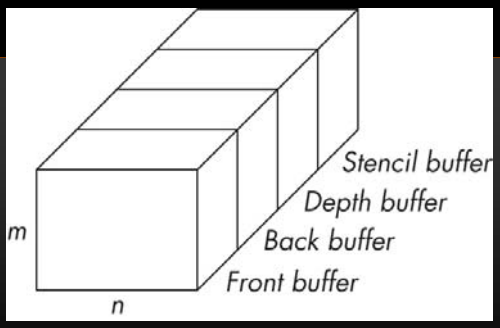
## Buffers

## Define a buffer by its spatial resolution (n x m) and its depth (or precision) k, the number of bits/pixel

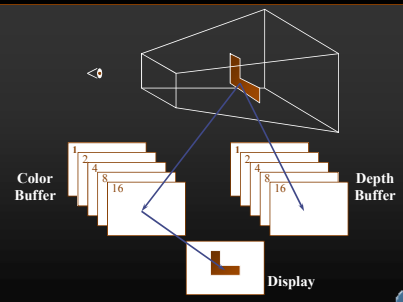






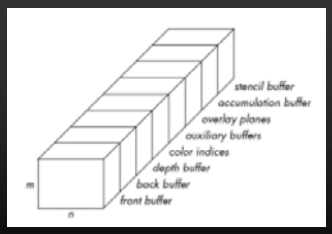

## Depth Buffering and Hidden Surface Removal



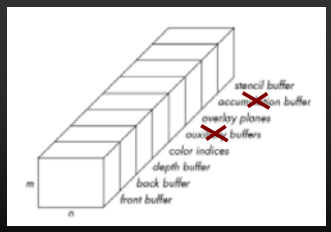

## Depth Buffering (already using it)

1. Hint for depth buffer resolution
```
void glfwWindowHint(GLFW_DEPTH_BITS, 16);
```
2. Enable depth buffering
```
glEnable( GL_DEPTH_TEST );
```
3. Clear color and depth buffers
```
glClear( GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT );
```
4. Render scene
5. Swap color buffers

## Other Buffers



## Other Buffers




## Using Framebuffers

- **clearing buffers**
  - clearing individual buffer is expensive
  - Use glClear with bitwise-ORed masks to clear multiple buffers
- **selecting color buffers for writing/clearing**
  - glBindFrameBuffer: useful in FBO (framebuffer object)

## Masking Buffers

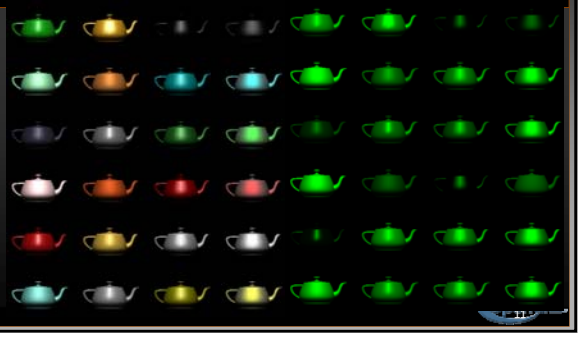
- **Before OpenGL writes data into the enabled color, depth, or stencil buffers, a masking operation is applied to the data, as specified with one of the following commands.**
- **A bitwise logical AND is performed with each mask and the corresponding data to be written**

## Masking Buffers (cont)

- **void glColorMask(GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha);**
- **void glDepthMask(GLboolean flag);**
- **void glStencilMask(GLuint mask);**
  - If a 1 appears in mask, the corresponding bit in the stencil buffer is written; where a 0 appears, the bit is not written.
- **The default values of all the GLboolean masks are GL_TRUE, and the default values for the two GLuint masks are all 1's**

**Red Mask GL_TRUE**
**Green Mask GL_TRUE**
**Blue Mask GL_TRUE**

**glColorMask(**
**GL_FALSE, GL_TRUE,**
**GL_FALSE, GL_FALSE)**
**Only Green Mask TRUE**



## Accumulation Buffer

- **Gone after OpenGL 3.1 (deprecated)**
- **Can use FBO for multi-pass rendering with an appropriate fragment program**
- **Useful for several effects**
- **Basically, same functions can be done with multi-pass rendering.**
- **Initially, it was the floating-point buffer but now all buffers can be floating-point!**

## Accessing Accumulation Buffer

`glAccum( op, value )`

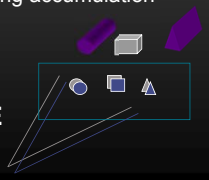
- operations
  - within the accumulation buffer: ***GL_ADD, GL_MULT***
  - from read buffer: ***GL_ACCUM, GL_LOAD***
  - transfer back to write buffer: ***GL_RETURN***
- **`glAccum(GL_ACCUM, 0.5)`** multiplies each value in write buffer by 0.5 and adds to accumulation buffer

## Accumulation Buffer Applications

- **Compositing**
- **Full Scene Antialiasing**
- **Depth of Field**
- **Filtering**
- **Motion Blur**

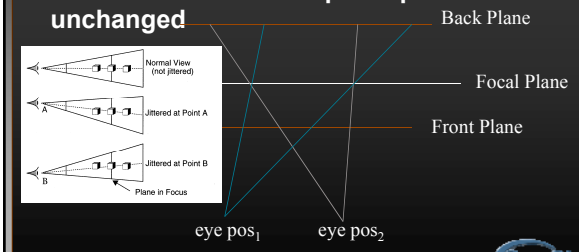
## Full Scene Antialiasing : *Jittering the view*

- **Each time we move the viewer, the image shifts**
  - Different aliasing artifacts in each image
  - Averaging images using accumulation buffer averages out these artifacts
- **Replaced with**
- **GL_MULTISAMPLE**

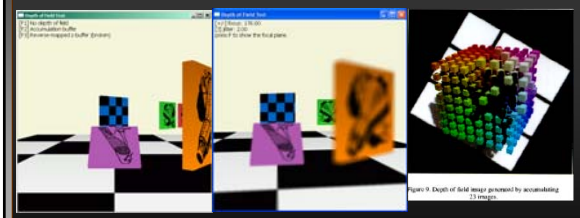


## Depth of Focus : *Keeping a Plane in Focus*

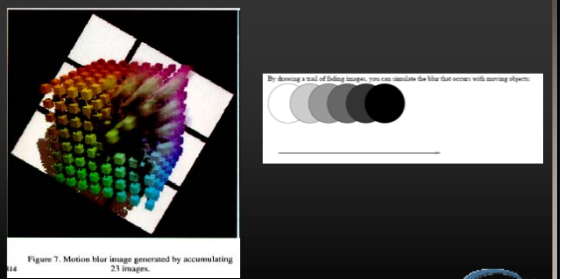
- **Jitter the viewer to keep one plane unchanged**



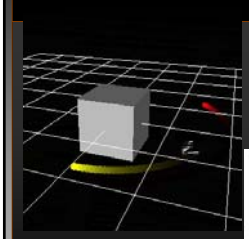

## Depth of Field



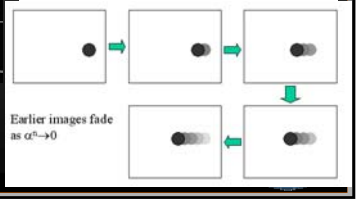
## Motion Blur



## Motion Blur w/o Accum.Buffer



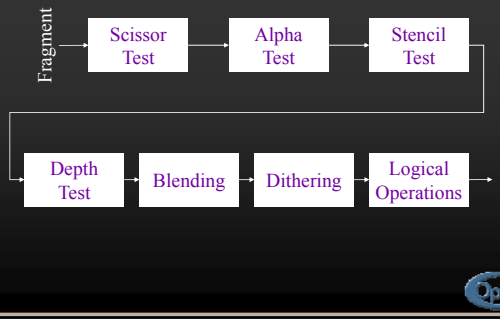
```
While (1) {
    render previous frame as background with α
    render current scene
    save result as next background
        [thus image containing previous frame]
}
```



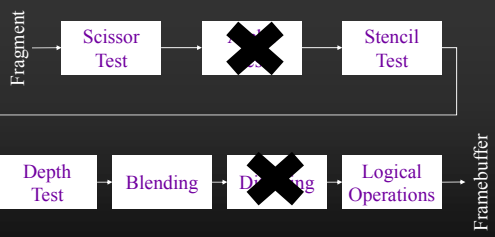
Earlier images fade as $\alpha^n \to 0$

**Details**:
scene dynamically render to texture; modulate with a polygon (1,1,1,a)

## Fragment Operations




## Fragment Operations




## Scissor Box

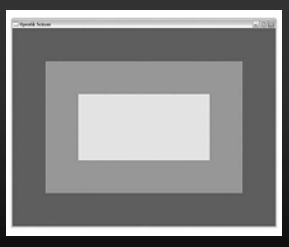
- **Additional Clipping Test**

```
glScissor( x, y, w, h )
```

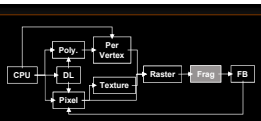
  - any fragments outside of box are clipped
  - useful for updating a small section of a viewport
    - affects `glClear()` operations

## Scissor test

```
void RenderScene(void)  {
        // Clear dark gray window
        glClearColor(0.2f, 0.2f, 1.2f, 0.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // Now set scissor to smaller gray sub region
        glClearColor(0.5f, 0.5f, 0.5f, 0.0f);
        glScissor(100, 100, 600, 400);
        glEnable(GL_SCISSOR_TEST);
        glClear(GL_COLOR_BUFFER_BIT);

        // Finally, an even smaller gray rectangle
        glClearColor(0.75f, 0.75f, 0.75f, 0.0f);
        glScissor(200, 200, 400, 200);
        glClear(GL_COLOR_BUFFER_BIT);

        // Turn scissor back off for next render
        glDisable(GL_SCISSOR_TEST);

        glutSwapBuffers();
}
```



## Alpha Test (deprecated)




- **Reject pixels based on their alpha value**

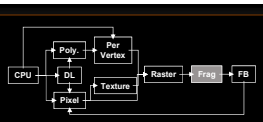
```
glAlphaFunc( func, value )
glEnable( GL_ALPHA_TEST )
```

  - use alpha as a mask in textures

- **Alpha test:**
  - accept/reject a fragment based on its alpha value
  - implement transparency
    - use this test to filter opaque objects
  - see-through decal (billboarding): reject the transparent fragments (from ruining the depth buffer)

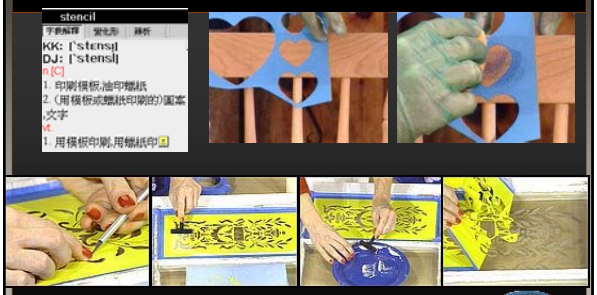
**Just use a fragment program!**

## Stencil Buffer




- Used to control drawing based on values in the stencil buffer
  - Fragments that fail the stencil test are not drawn
  - Example: create a mask in stencil buffer and draw only objects not in mask area

## Stenciling




## Mimicking Stencil

- Compose stencil template
- Control template then render
- Multi-pass rendering

silhouette

## Controlling Stencil Buffer

`glStencilFunc(func,ref,mask)`

- compare value in buffer with `ref` using `func`
- only applied for bits in `mask` which are 1
- `func` is one of standard comparison functions

`glStencilOp(fail,zfail,zpass)`

- Allows changes in stencil buffer based on passing or failing stencil and depth tests: `GL_KEEP`, `GL_INCR`
- `glStencilFuncSeparate(face,ref,mask)`
- `glStencilOpSeparate(face,fail,zfail,zpass)`

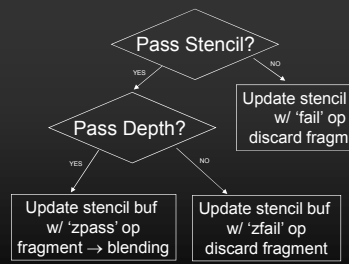
## glStencilFunc(func, ref, mask)

never
always
<
<=
=
>=
>
!=

Compare value in stencil buff with ref using func

Bit-wise mask for comparison

## glStencilOp(fail,zfail,zpass)




- Keep
- Zero
- Replace
- Incr (_WRAP)
- Decr (_WRAP)
- Invert

## How to set the stencil?

## Creating a Mask

```
glfwWindowHint(GLFW_STENCIL_BITS, 16);
glEnable( GL_STENCIL_TEST );
glClearStencil( 0x0 );

glStencilFunc( GL_ALWAYS, 0x1, 0x1 );
glStencilOp( GL_REPLACE, GL_REPLACE,
             GL_REPLACE );
```

- *draw mask*

## Using Stencil Mask

```
glStencilFunc( GL_EQUAL, 0x1, 0x1 )
```

- draw objects where stencil = 1

```
glStencilFunc( GL_NOT_EQUAL, 0x1, 0x1 );
glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );
```

- draw objects where stencil != 1

## Example: Room w/ Window

## Room with a view

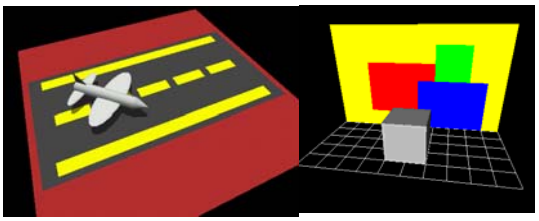
1. Turn off color buffer
2. Turn off depth buffer updates
3. Turn on stencil buffer
4. Setup the stencil test
5. Draw the window
6. Sets up the stencil test for background
7. Turn on the color buffer
8. Turn on the depth buffer
9. Draw the background
10. Setup test for the wall
11. Draw the wall
12. Reset state
13. Draw any interior

## Room with a view

1. Turn off color buffer
2. Turn off depth buffer updates
3. Turn on stencil buffer
4. Setup the stencil test
5. Draw the window
6. Sets up the stencil test for background
7. Turn on the color buffer
8. Turn on the depth buffer
9. Draw the background
10. Setup test for the wall
11. Draw the wall
12. Reset state
13. Draw any interior

1. glColorMask(F,F,F,F)
2. glDepthMask(F)
3. glEnable(stencil-test)
4. glStencilFunc(A,0x01,0x01) glStencilOp(K,K,R)
5. *Draw the window*
6. glStencilFunc(=,0x01, 0x01) glStencilOp(k,k,k)
7. glColorMask(T,T,T,T)
8. glDepthMask(T)
9. *Draw background*
10. glStencilFunc(!=,0x01, 0x01)
11. *Draw wall*
12. glDisable(stencil-test)
13. *Draw anything else*

## Decal



Bad way to resolve z-fighting

37

## Decaling w/ Depth Buffer (Painter's Alg)

1. Disable depth buffer updates
2. Draw the base polygon
3. Draw the decal polygons
4. Disable color buffer updates
5. Enable depth buffer updates
6. Draw base polygon
7. Reset state (enable color buffers)

## Decaling w/ Depth Buffer (Painter's Alg)

1. Disable depth buffer updates glEnable(GL_DEPTH_TEST) glDepthMask(GL_FALSE)
2. Draw the base polygon
3. Draw the decal polygons
4. Disable color buffer updates glColorMask(GL_FALSE,…)
5. Enable depth buffer updates glDepthMask(GL_TRUE)
6. Draw base polygon
7. Reset state (enable color buffers) glColorMask(GL_TRUE…)

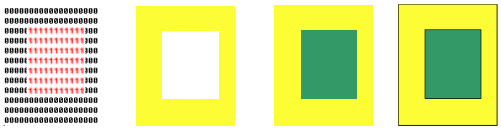
## Decaling w/ stencil buffer

A. Create a mask in the stencil buffer which defines the decal region

B. Use this mask in 2 passes:
   base polygon
   decal polygon(s)

## Stenciling

- Steps to draw 2 coplanar rectangles:
  1. Make the stencil for yellow one first (by drawing the green polygon)
  2. Draw the yellow one with the stencil
  3. Draw the green one

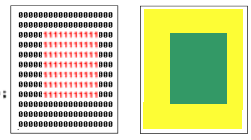


41

## Stenciling (cont)

```
glEnable(GL_STENCIL_TEST);

glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT |
        GL_STENCIL_BUFFER_BIT);
// so that all pixels in stencil buffer are 0

// MAKING THE STENCIL:
// disable write to color buffer
glColorMask (GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDisable(GL_DEPTH_TEST)
glStencilFunc (GL_ALWAYS, 0x1, 0x0);
glStencilOp (GL_REPLACE, GL_REPLACE, GL_REPLACE);

// [draw GREEN rectangle], to the area of GREEN filled with 1s

// ready to write to color buffer
glColorMask (GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);

// first draw YELLOW rectangle to 0s
glStencilFunc (GL_EQUAL, 0x0, 0x1);
// no change to the stencil values
glStencilOp (GL_KEEP, GL_KEEP, GL_KEEP);

// [draw YELLOW rectangle]
// disable stencil test
glDisable (GL_STENCIL_TEST);

// [draw GREEN rectangle]
glEnable(GL_DEPTH_TEST)
```

Stencil buffer Color buffer



42

## Decaling w/ stencil buffer

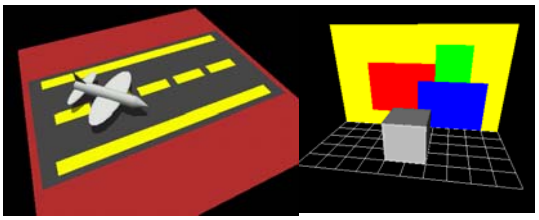
1. Enable stenciling
2. Set test to always pass
   w/ref=1, mask=1
3. Set stencil op
   1: if depth passes
   0: if depth fails
4. Draw the base polygon
5. Set stencil function to pass
6. Disable writes to the stencil buf
7. Turn off depth buffering
8. Render the decal polygon

## Decaling w/ stencil buffer

1. Enable stenciling
2. Set test to always pass
   w/ref=1, mask=1
3. Set stencil op
   1: if depth passes
   0: if depth fails
4. Draw the base polygon
5. Set stencil function to pass
6. Disable writes to the stencil buf
7. Turn off depth buffering
8. Render the decal polygon
9. Reset state

```
glEnable(GL_Stencil_Test)
glStencilFunc(GL_ALWAYS,1,1)

glStencilOp(GL_KEEP, GL_ZERO,
   GL_REPLACE)

glStencilFunc(GL_EQUAL,1,1)
glStencilOp(GL_KEEP, GL_KEEP,
   GL_KEEP)
glDisable(GL_DEPTH_TEST)

glDisable(GL_STENCIL_TEST)
glEnable(GL_DEPTH_TEST)
```

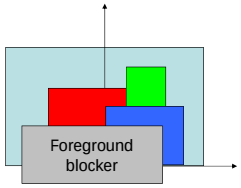
## Decal



How to resolve z-fighting

45

## Decaling




[0] Base: (-5,0)→(5,6)
[1] R: (-3,1)→(2,4)
[2] G: (1,2)→(3,5)
[3] B: (0,0)→(4,3)

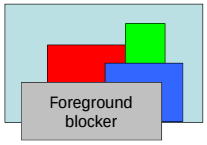
1. Draw everything else (set up the depth buffer)
2. Draw base; set the blocker id to be 8; rest 0
3. Arrange the decal id in increasing order
4. The decals can be drawn in any order

Test the program

46

```
glEnable (GL_LIGHTING);
glPushMatrix();
    glTranslatef (0.,1,3.);
    glutSolidCube (2.0);
glPopMatrix();
glDisable (GL_LIGHTING);
glEnable (GL_STENCIL_TEST);

glStencilFunc (GL_ALWAYS, 8, 0xff);
glStencilOp (GL_KEEP, GL_REPLACE, GL_ZERO);
glColor3ub (255,255,0); glRectf (-5,0, 5,6);

glDisable (GL_DEPTH_TEST);
glStencilFunc (GL_GEQUAL, 1, 0xff);
glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE);
glColor3ub (255,0,0);   glRectf (-3,1, 2,4);

glStencilFunc (GL_GEQUAL, 2, 0xff);
glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE);
glColor3ub (0,255,0);   glRectf (1,2, 3,5);

glStencilFunc (GL_GEQUAL, 3, 0xff);
glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE);
glColor3ub (0,0,255);   glRectf (0,0, 4,3);

glDisable (GL_STENCIL_TEST);
glEnable (GL_DEPTH_TEST);  // restore
```

**StencilOp(fail, zfail, zpass)**




Color buffer

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | |
| 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 0 | 0 | |
| 0 | 0 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 0 | |
| 0 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 3 | 0 | |
| 0 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 3 | 0 | |

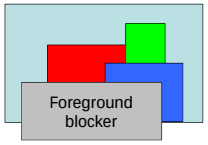
Stencil buffer

**What is wrong with this?**

47

```
glEnable (GL_LIGHTING);
glPushMatrix();
    glTranslatef (0.,1,3.);
    glutSolidCube (2.0);
glPopMatrix();
glDisable (GL_LIGHTING);
glEnable (GL_STENCIL_TEST);

glStencilFunc (GL_ALWAYS, 8, 0xff);
glStencilOp (GL_KEEP, GL_REPLACE, GL_ZERO);
glColor3ub (255,255,0); glRectf (-5,0, 5,6);

glDisable (GL_DEPTH_TEST);
glStencilFunc (GL_GEQUAL, 1, 0xff);
glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE);
glColor3ub (255,0,0);   glRectf (-3,1, 2,4);

glStencilFunc (GL_GEQUAL, 2, 0xff);
glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE);
glColor3ub (0,255,0);   glRectf (1,2, 3,5);

glStencilFunc (GL_GEQUAL, 3, 0xff);
glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE);
glColor3ub (0,0,255);   glRectf (0,0, 4,3);

glDisable (GL_STENCIL_TEST);
glEnable (GL_DEPTH_TEST);  // restore
```

**StencilOp(fail, zfail, zpass)**




Color buffer

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | |
| 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 0 | 0 | |
| 0 | 0 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 0 | |
| 0 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 3 | 0 | |
| 0 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 3 | 0 | |

Stencil buffer

Drawn in different order!

**What is wrong with this?**

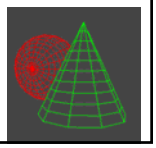
48

## Hidden Lines

-polygon offset, draw twice

Polygon Offset (depth-buffer biasing)

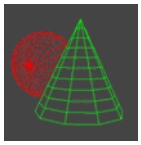
```
glEnable(GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
set_color(foreground);
draw_object_with_filled_polygons();
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(1.0, 1.0);
set_color(background);
draw_object_with_filled_polygons();
glDisable(GL_POLYGON_OFFSET_FILL);
```



## Hidden Lines

draw on per object basis with stencilling

```
glEnable(GL_STENCIL_TEST);
glEnable(GL_DEPTH_TEST);
glClear(GL_STENCIL_BUFFER_BIT);
glStencilFunc(GL_ALWAYS, 0, 1);
glStencilOp(GL_INVERT, GL_INVERT, GL_INVERT);
set_color(foreground);
for (i=0; i < max; i++) {
    outline_polygon(i);
    set_color(background);
    glStencilFunc(GL_EQUAL, 0, 1);
    glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
    fill_polygon(i);
    set_color(foreground);
    glStencilFunc(GL_ALWAYS, 0, 1);
    glStencilOp(GL_INVERT, GL_INVERT, GL_INVERT);
    outline_polygon(i);
}
```

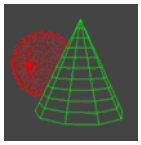


## draw on per object basis with stencilling

- Outline polygon (FG) setting the stencil
  - **glStencilFunc(GL_ALWAYS, 0, 0x1)**
  - **GLStencilOp(GL_INVERT, GL_INVERT, GL_INVERT)**
  - **Set color to foreground**
  - **Draw the polygon outline**
- Fill polygon (BG) where stencil is not set
  - **glStencilFunc(GL_EQUAL,0,0x1)**
  - **glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP)**
  - **Fill the polygon (BG)**
- Outline polygon (FG) resetting stencil
  - **glStencilFunc(GL_ALWAYS, 0, 0x1)**
  - **GLStencilOp(GL_INVERT, GL_INVERT, GL_INVERT)**
  - **Set color to foreground**
  - **Draw the polygon outline**

## Hidden Lines

Correct method – save the depth buffer

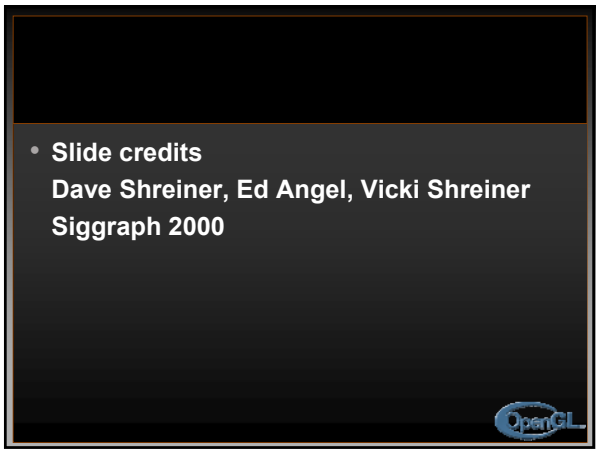


## Correct Version

- Need to save/reset the depth-buffer for each object.
- See the web-page (Lectures notes) for the details

## Silhouettes

- See web-page (lectures notes) solutions


• Slide credits
Dave Shreiner, Ed Angel, Vicki Shreiner
Siggraph 2000
OpenGL