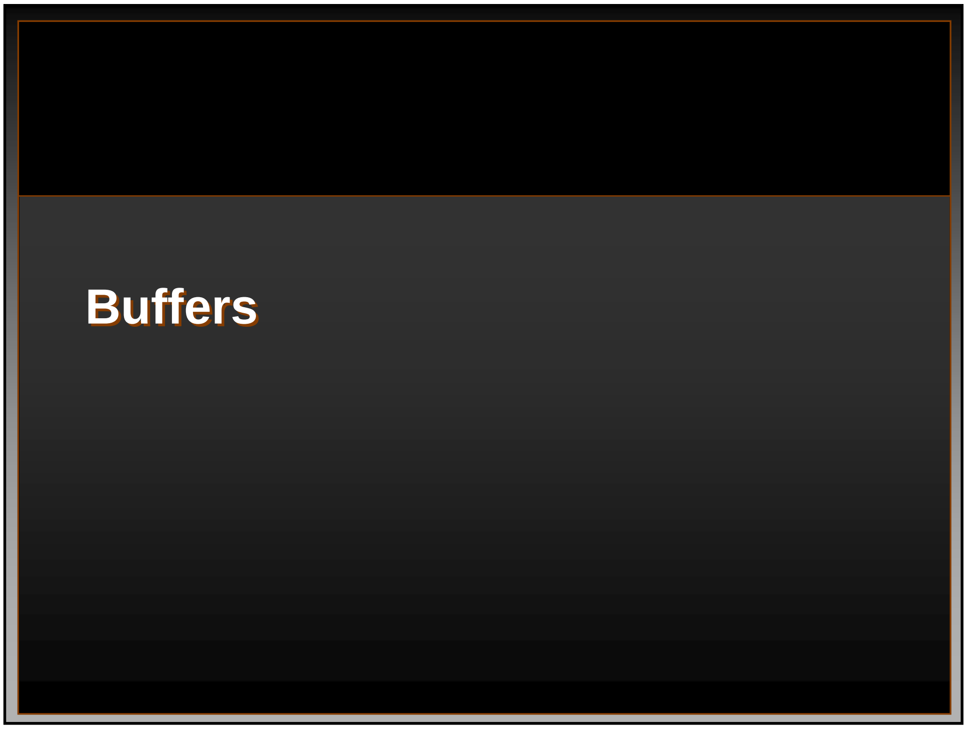
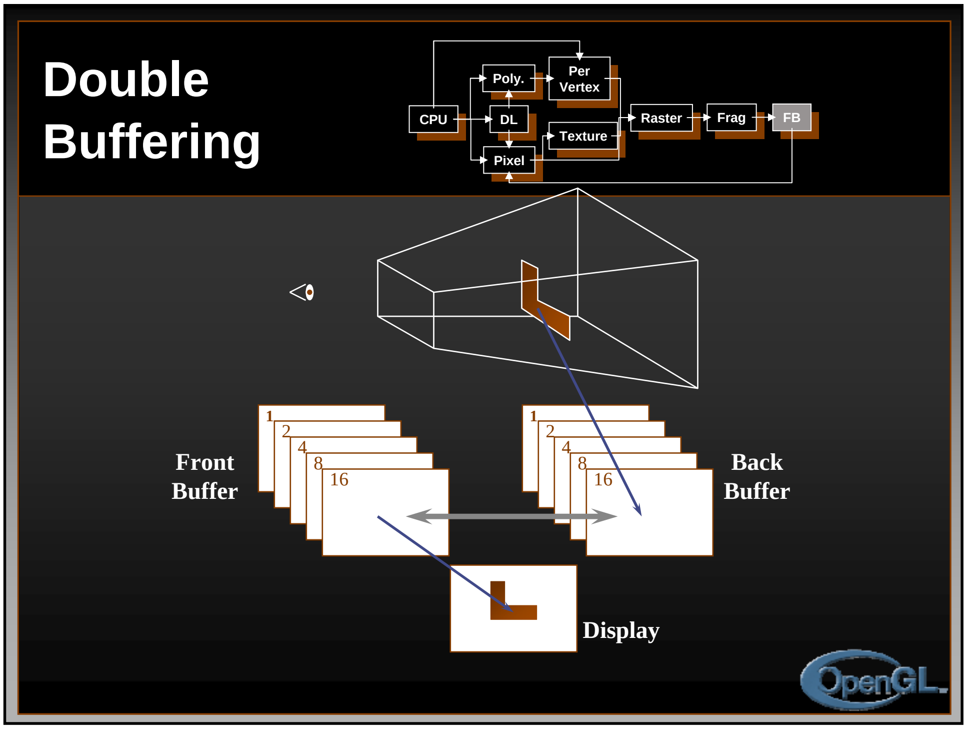
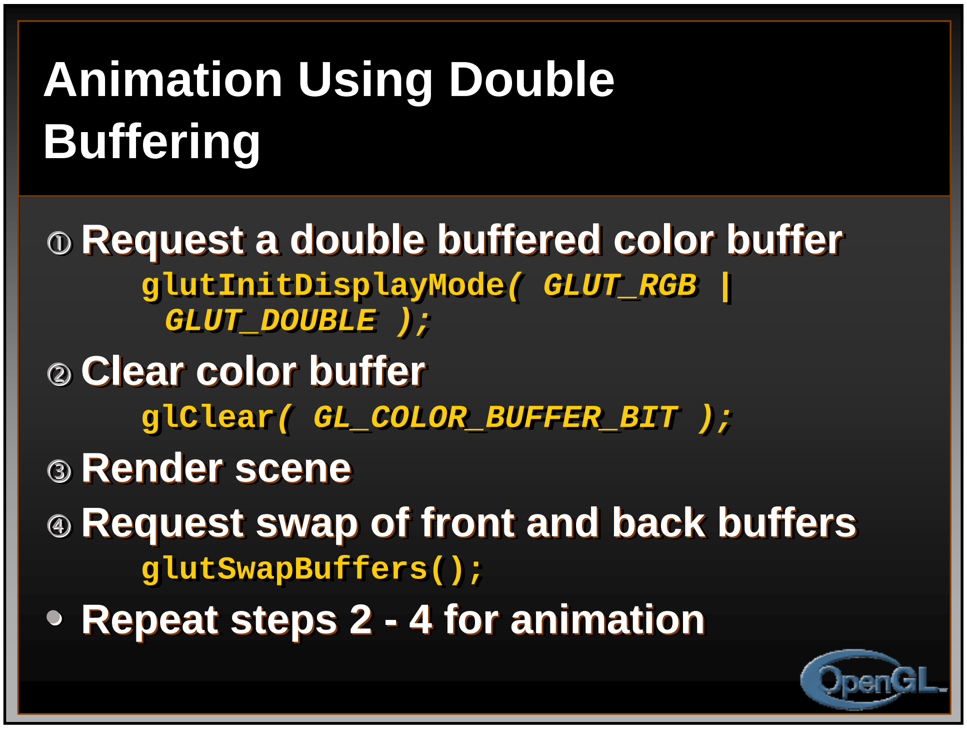


Buffers



Double Buffering



Animation Using Double Buffering

① Request a double buffered color buffer

```
glutInitDisplayMode( GLUT_RGB |
  GLUT_DOUBLE );
```

② Clear color buffer

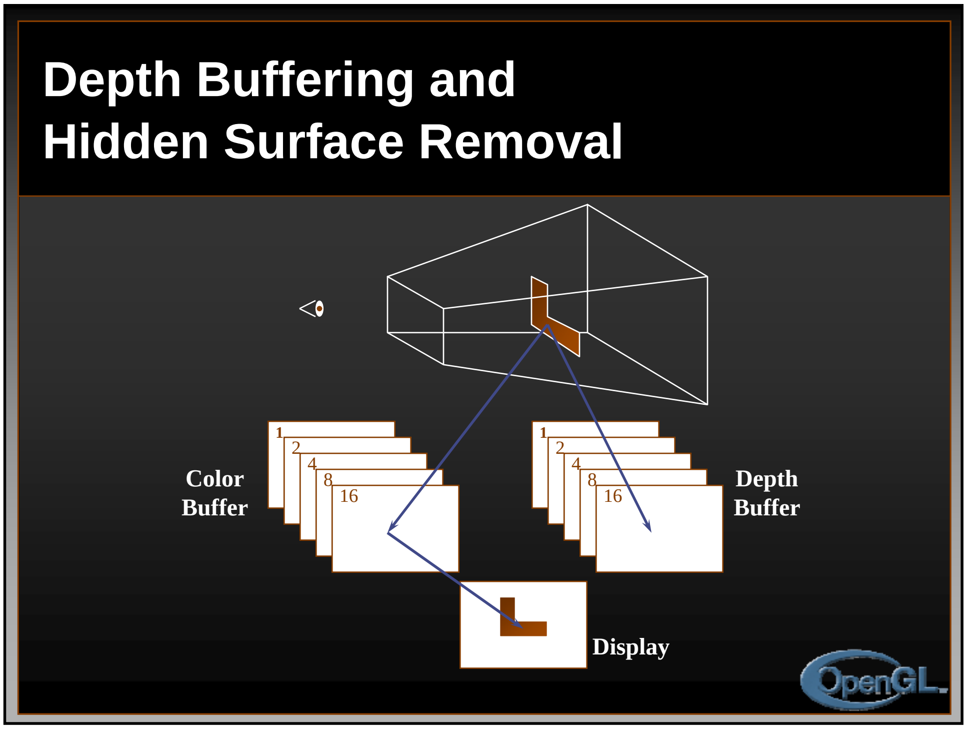
```
glClear( GL_COLOR_BUFFER_BIT );
```

③ Render scene

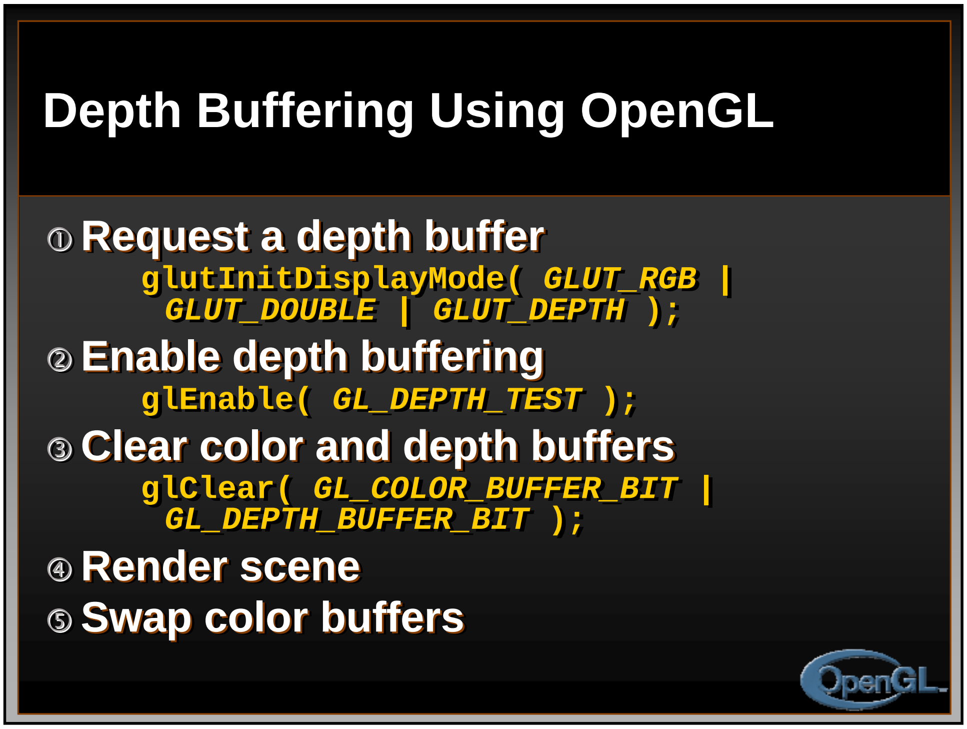
④ Request swap of front and back buffers

```
glutSwapBuffers();
```

- Repeat steps 2 - 4 for animation



Depth Buffering and Hidden Surface Removal



Depth Buffering Using OpenGL

① Request a depth buffer

```
glutInitDisplayMode( GLUT_RGB |
  GLUT_DOUBLE | GLUT_DEPTH );
```

② Enable depth buffering

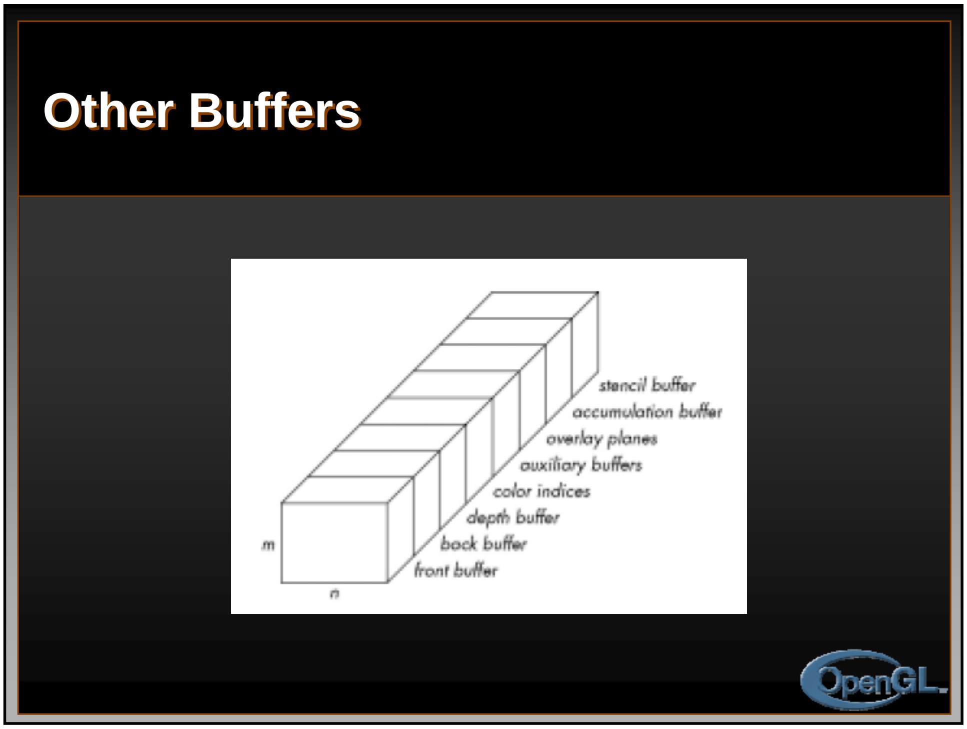
```
glEnable( GL_DEPTH_TEST );
```

③ Clear color and depth buffers

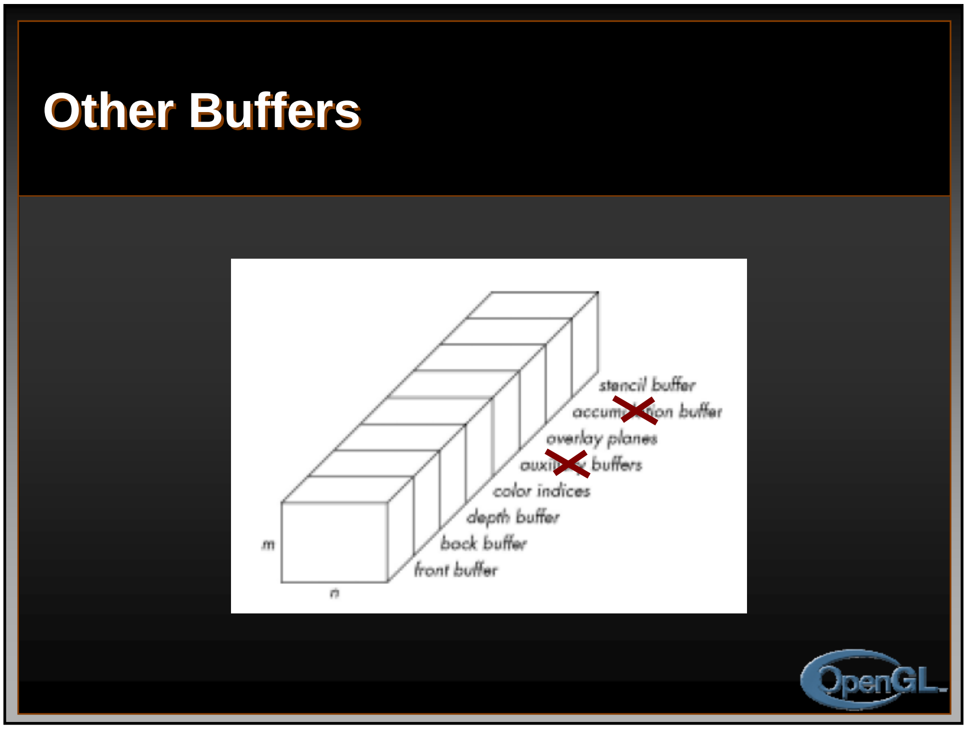
```
glClear( GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT );
```

④ Render scene

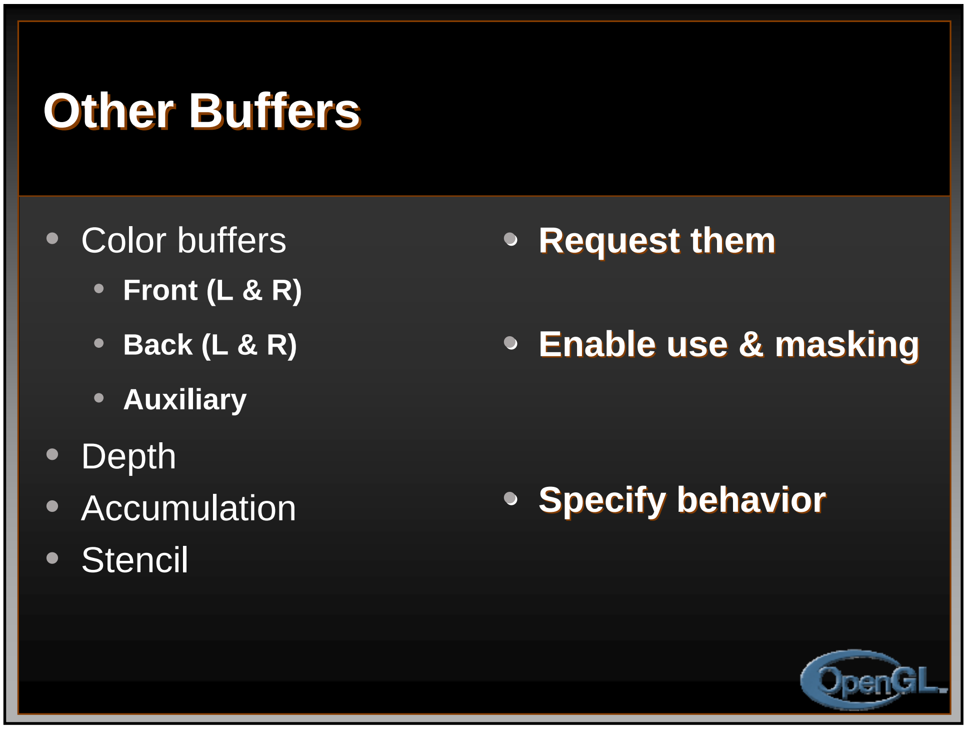
⑤ Swap color buffers



Other Buffers



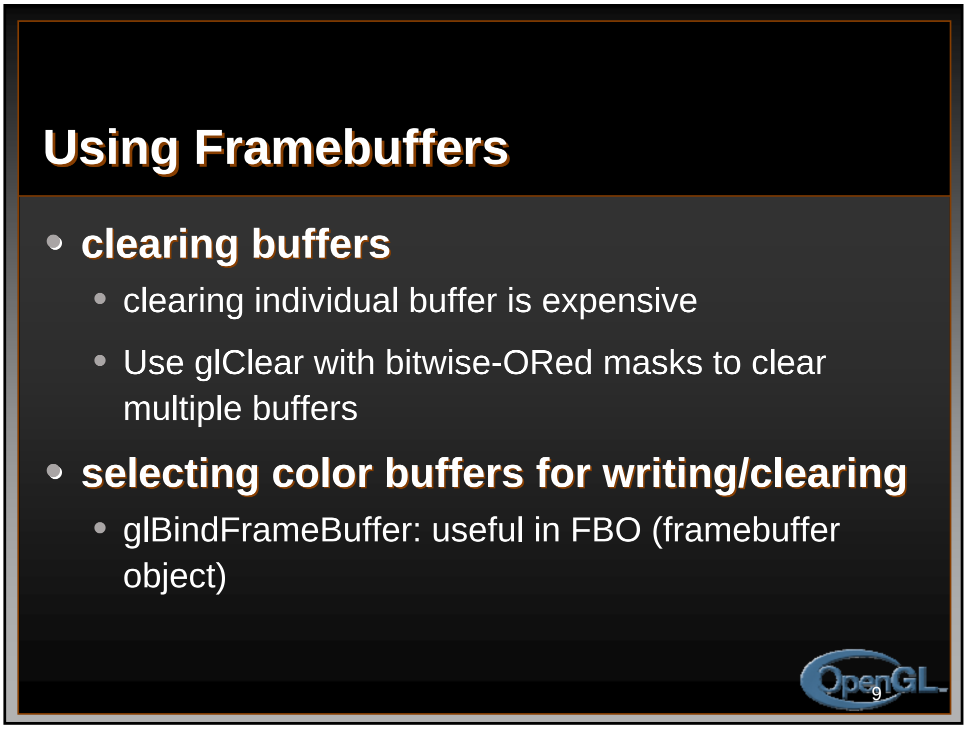
Other Buffers



Other Buffers

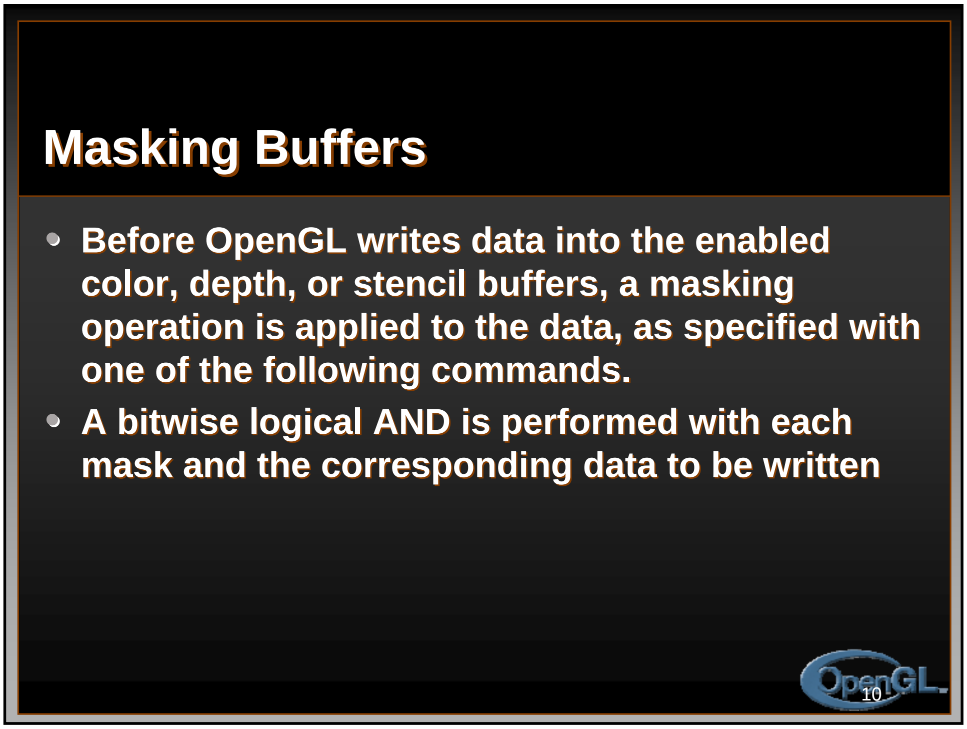
- Color buffers
  - **Front (L & R)**
  - **Back (L & R)**
  - **Auxiliary**
- Depth
- Accumulation
- Stencil

- **Request them**
- **Enable use & masking**
- **Specify behavior**



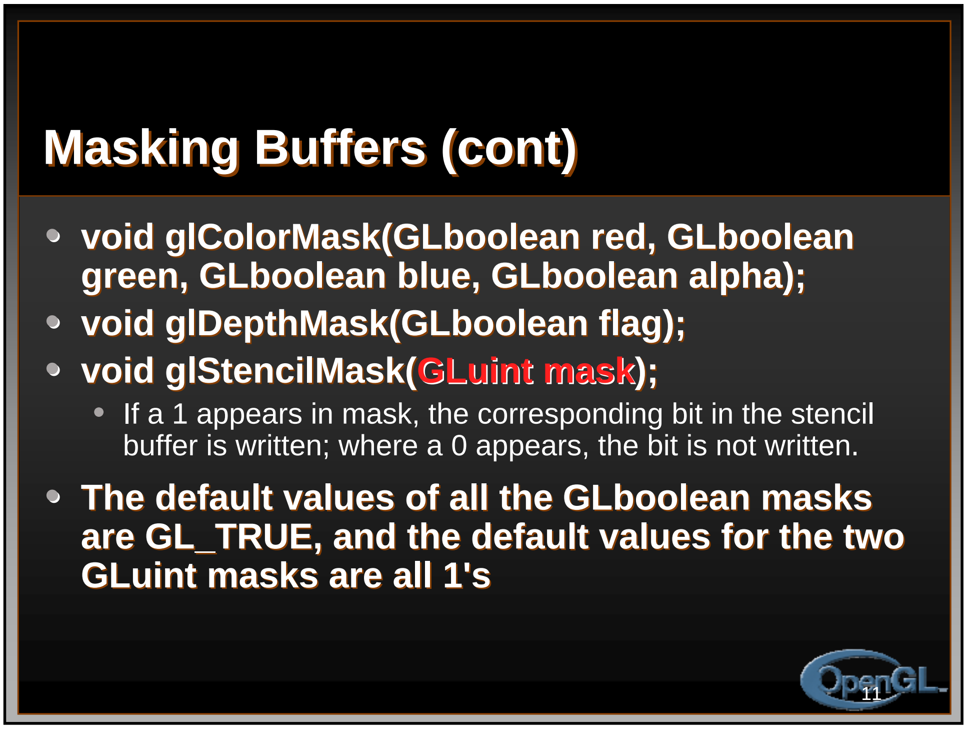
Using Framebuffers

- **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)



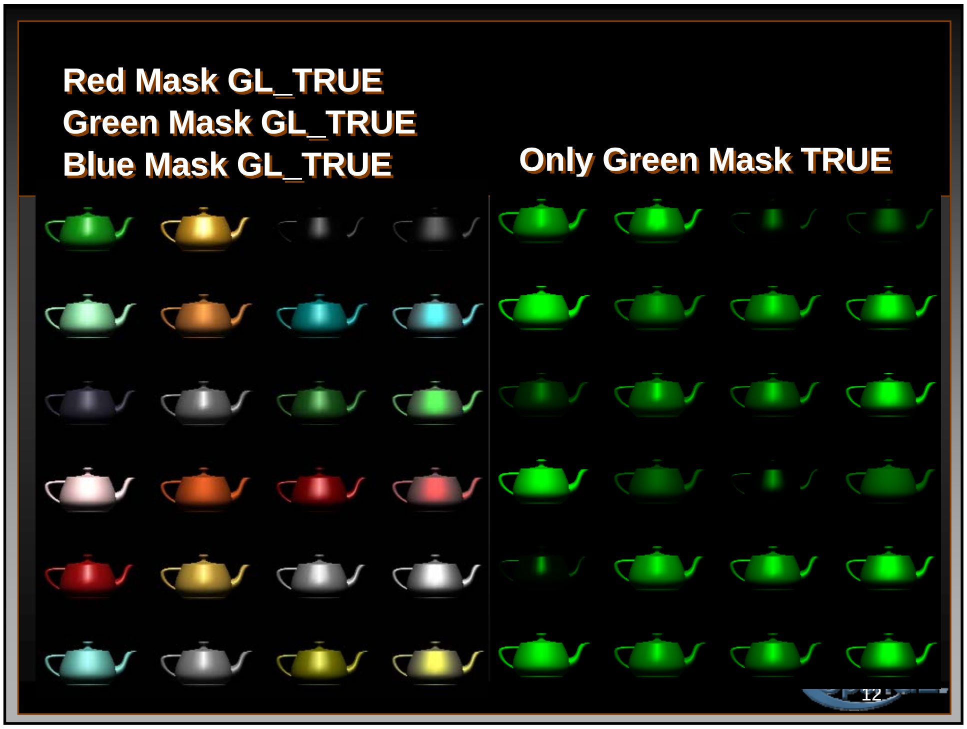
Masking Buffers

- **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**



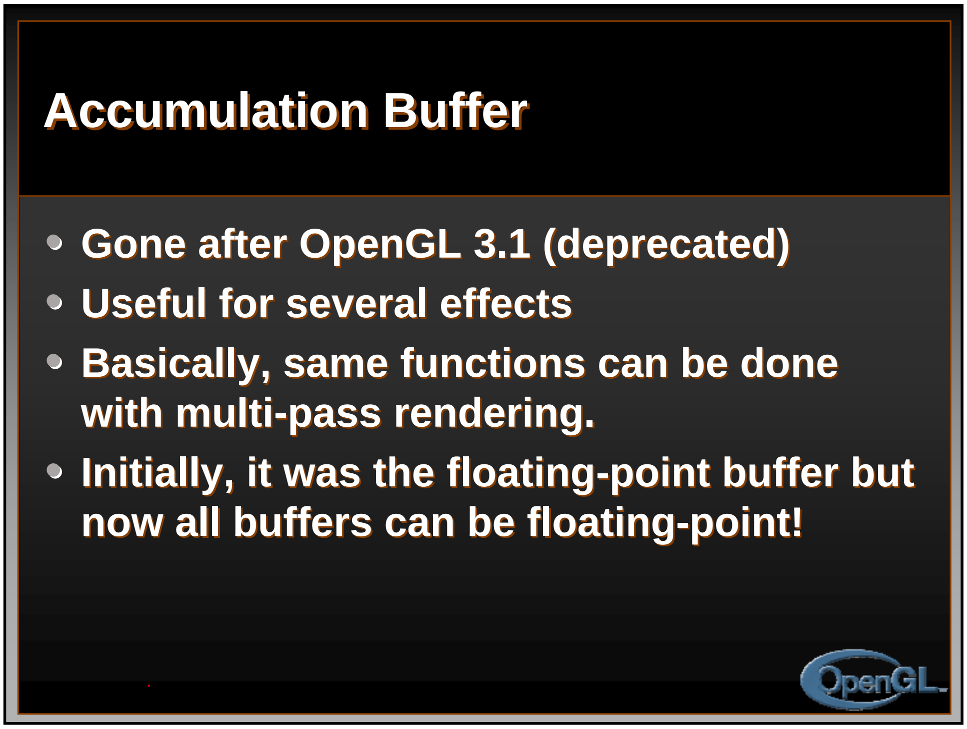
Masking Buffers (cont)

- 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



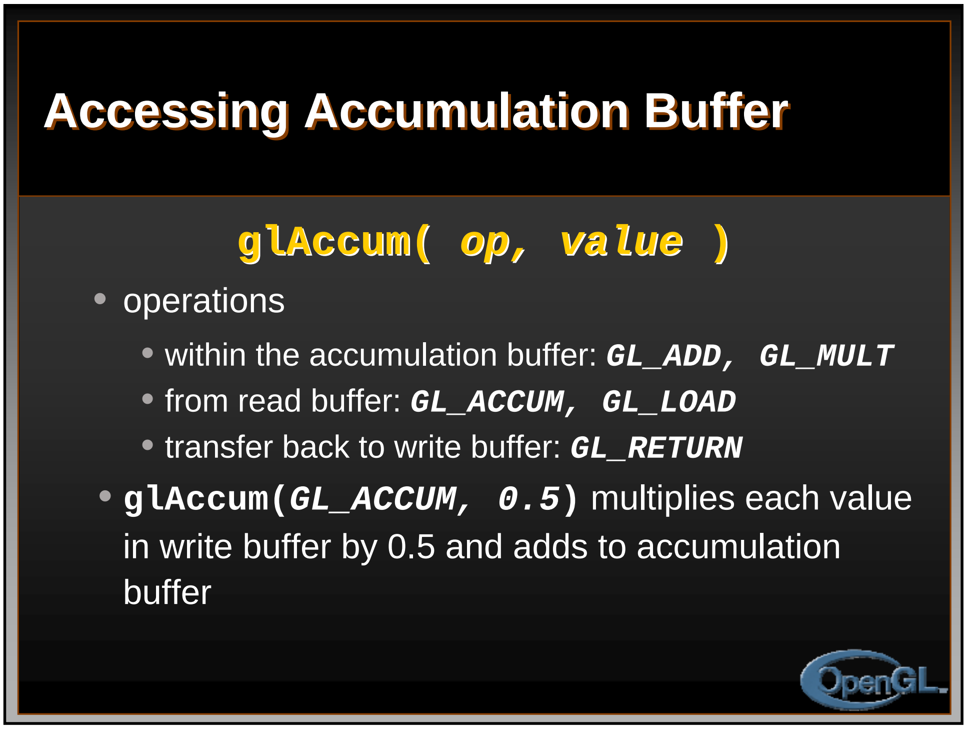
Red Mask GL_TRUE
Green Mask GL_TRUE
Blue Mask GL_TRUE

Only Green Mask TRUE



Accumulation Buffer

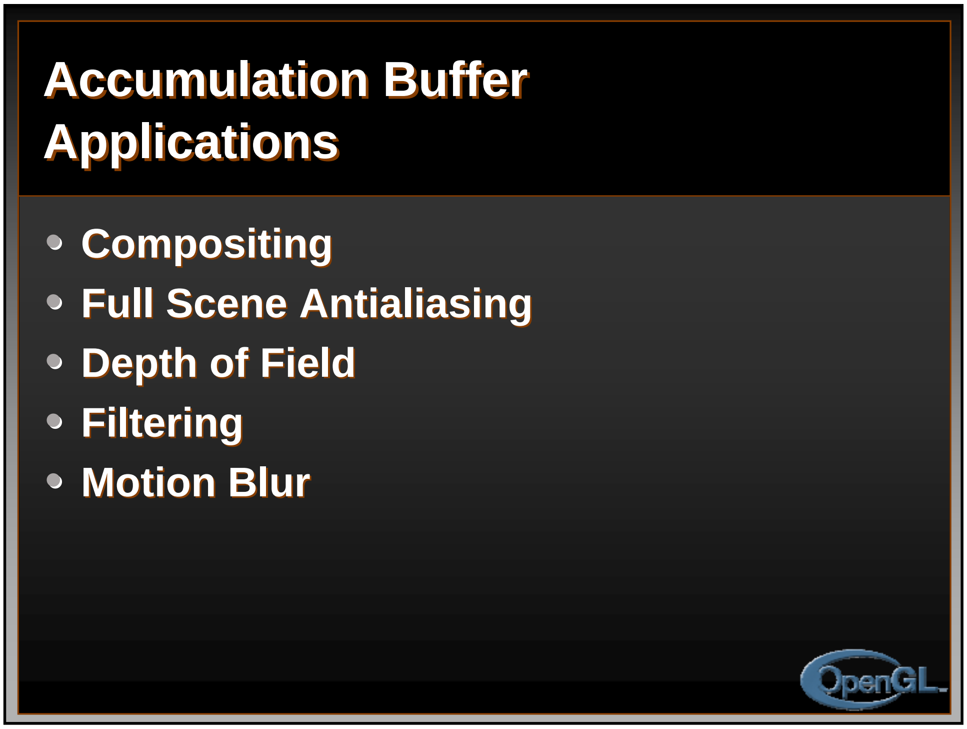
- Gone after OpenGL 3.1 (deprecated)
- Useful for several effects
- Basically, same functions can be done with multi-pass rendering.
- Initially, it was the floating-point buffer but now all buffers can be floating-point!



Accessing Accumulation Buffer

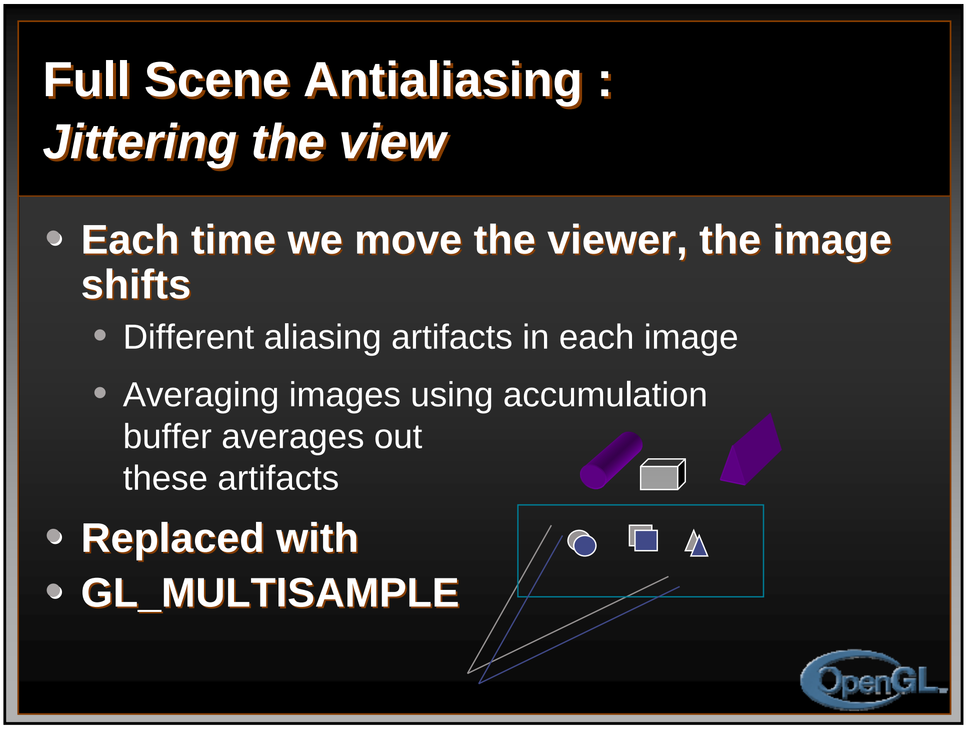
`glAccum( op, value )`

- 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

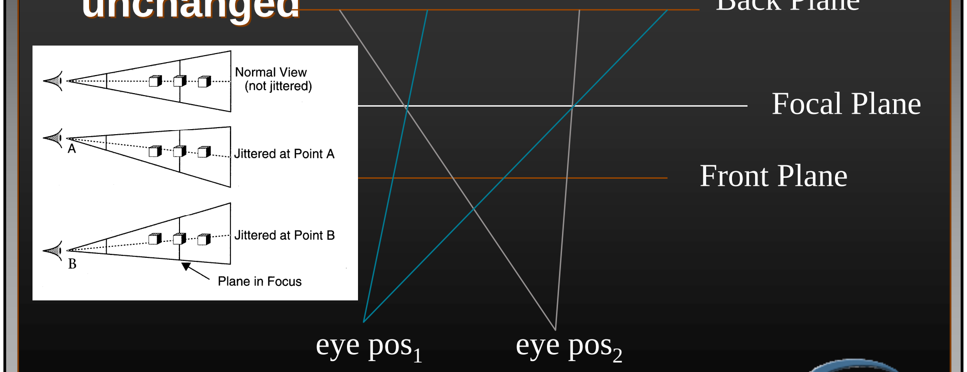


Full Scene Antialiasing : *Jittering the view*

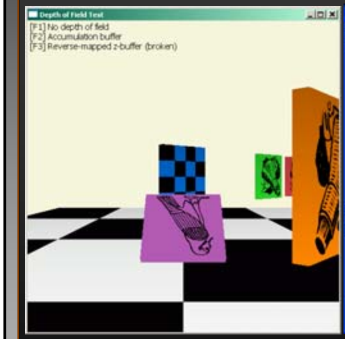
- Each time we move the viewer, the image shifts
  - Different aliasing artifacts in each image
  - Averaging images using accumulation buffer averages out these artifacts
- Replaced with
- GL_MULTISAMPLE

Depth of Focus : *Keeping a Plane in Focus*

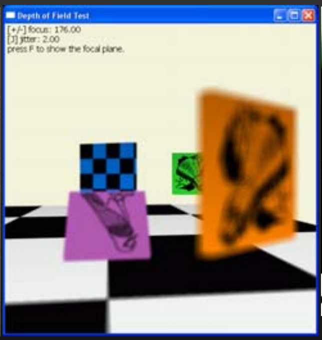
- Jitter the viewer to keep one plane unchanged



Depth of Field



[F1] No depth of field
[F2] Accumulation buffer
[F3] Reverse-mapped z-buffer (broken)



[+/-] focus: 176.00
[J] jitter: 2.00
press F to show the focal plane.

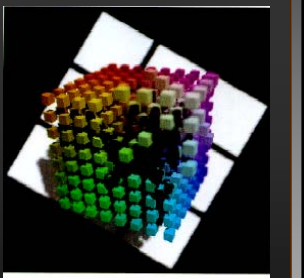
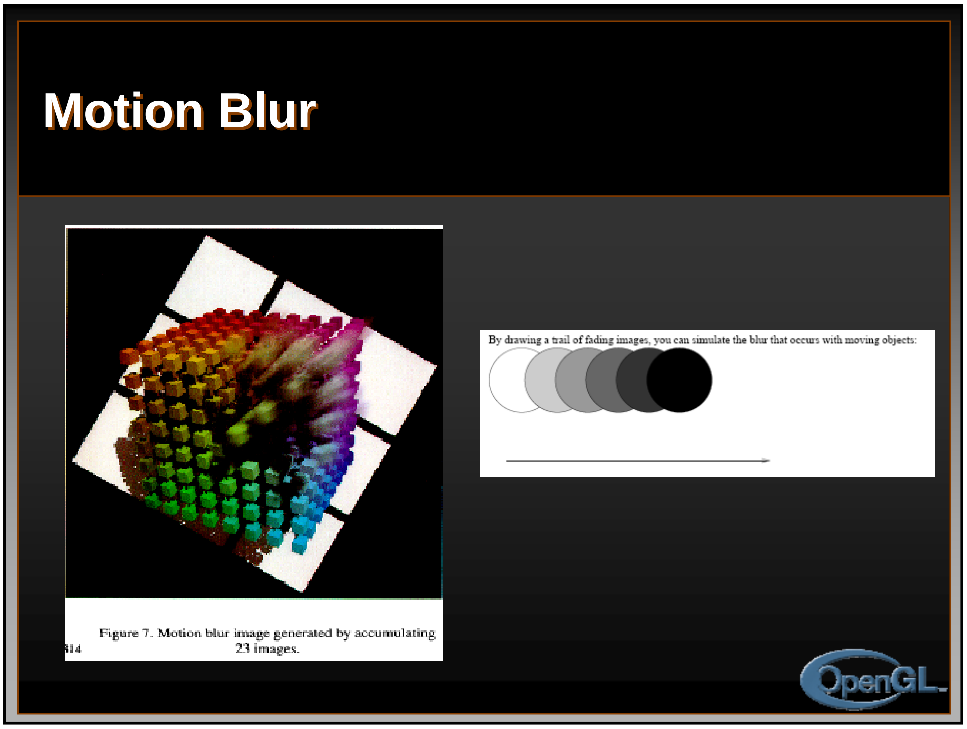


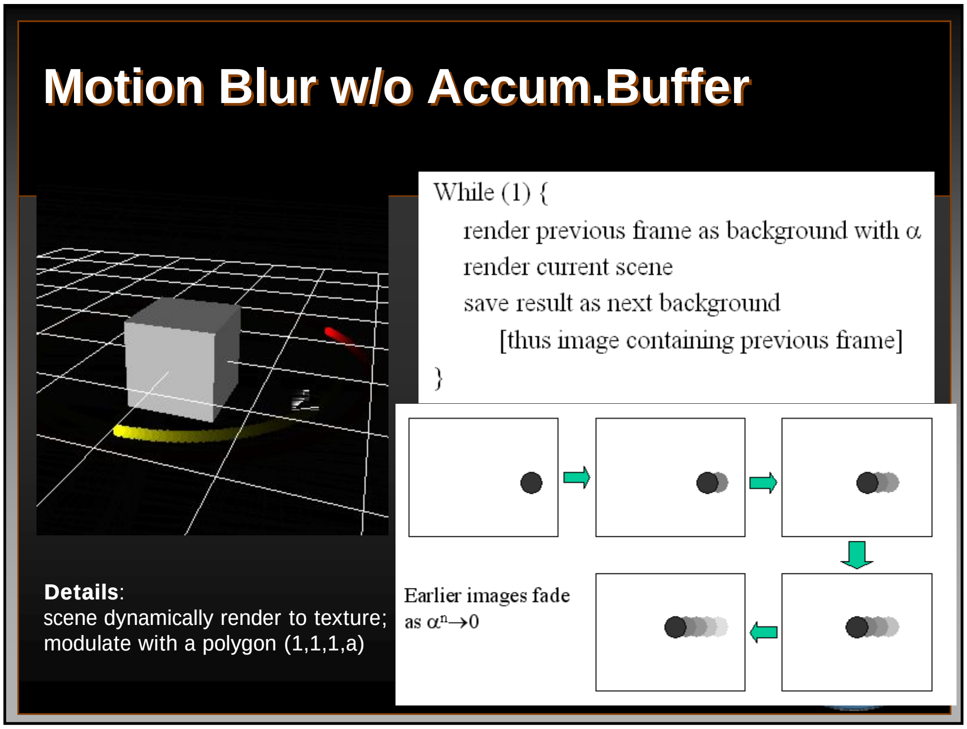
Figure 9. Depth of field image generated by accumulating 23 images.



Motion Blur

Figure 7. Motion blur image generated by accumulating 23 images.

By drawing a trail of fading images, you can simulate the blur that occurs with moving objects:

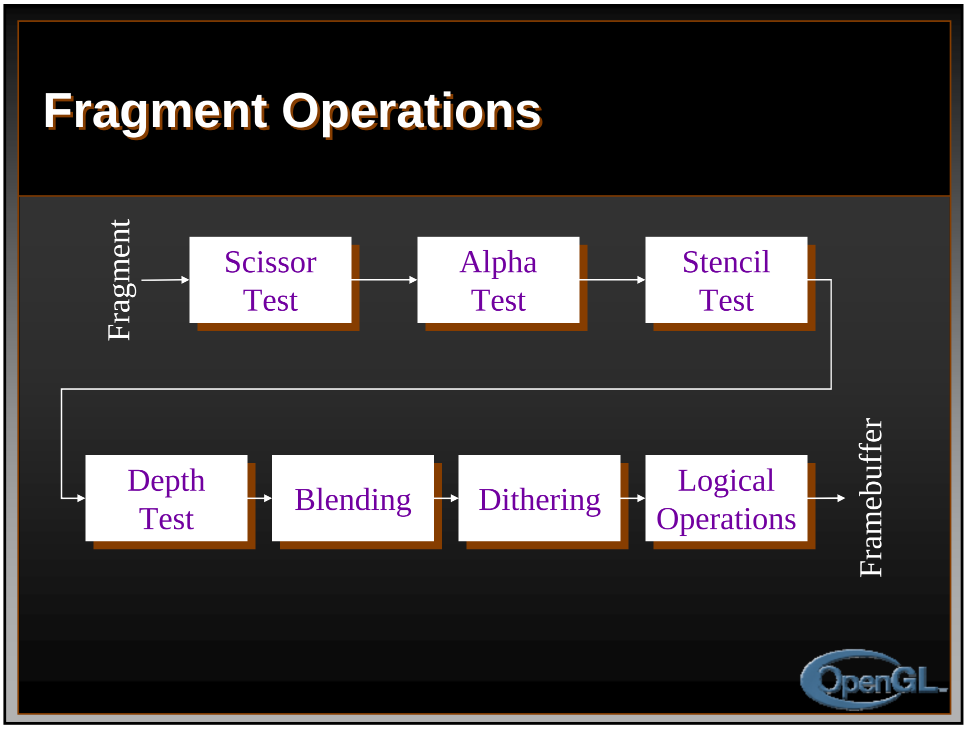


Motion Blur w/o Accum.Buffer

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

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

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



Fragment Operations

Fragment

Scissor Test

Alpha Test

Stencil Test

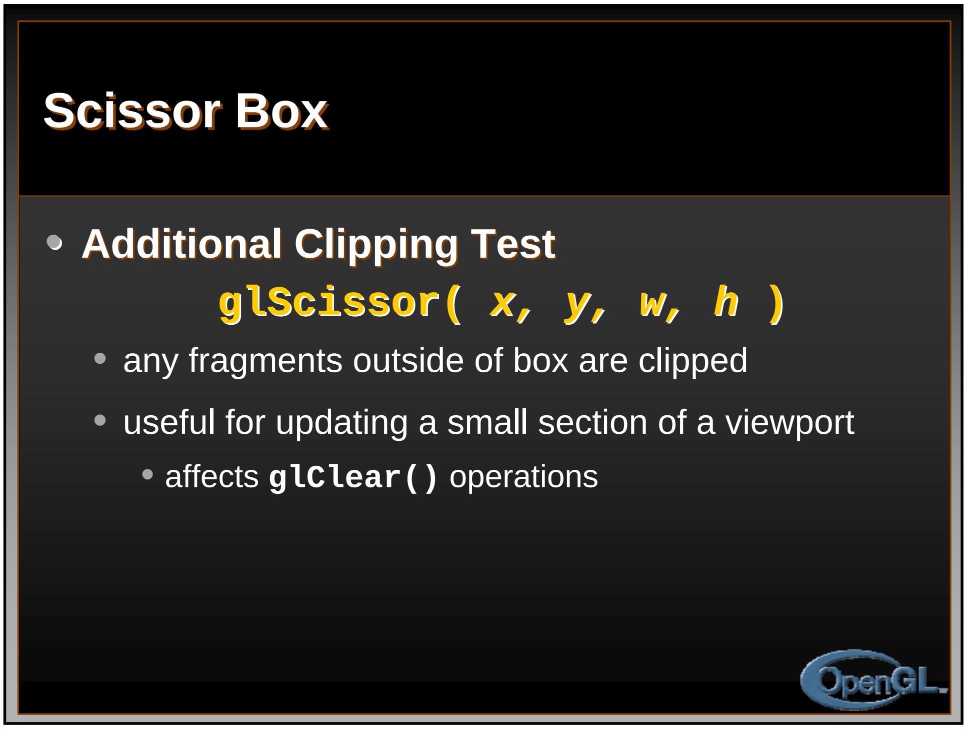
Depth Test

Blending

Dithering

Logical Operations

Framebuffer

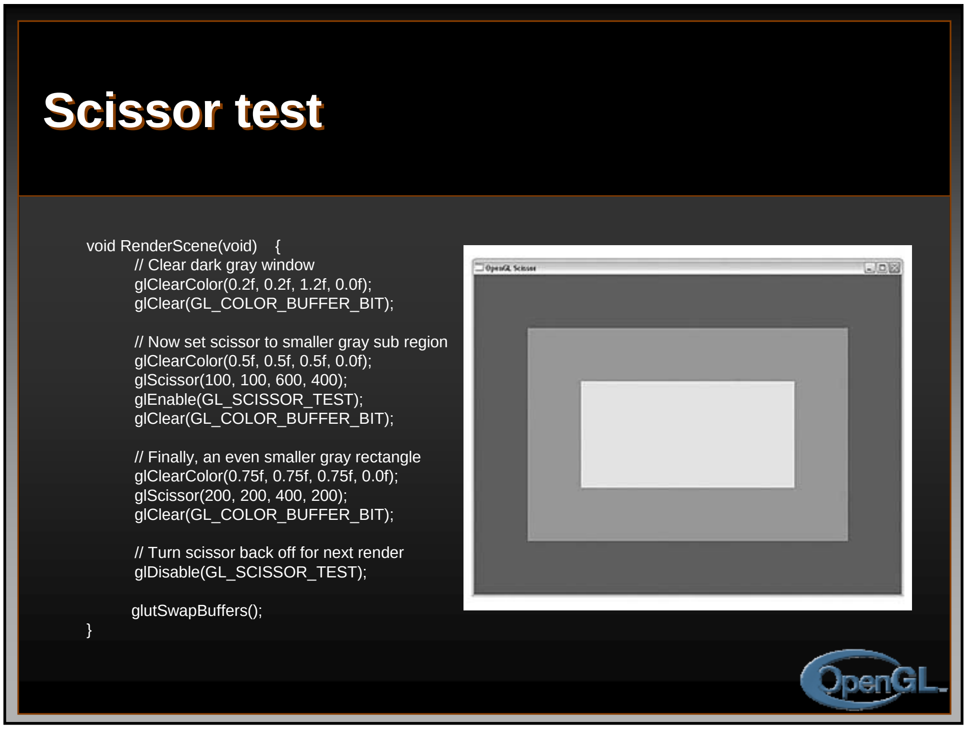


Scissor Box

- **Additional Clipping Test**

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

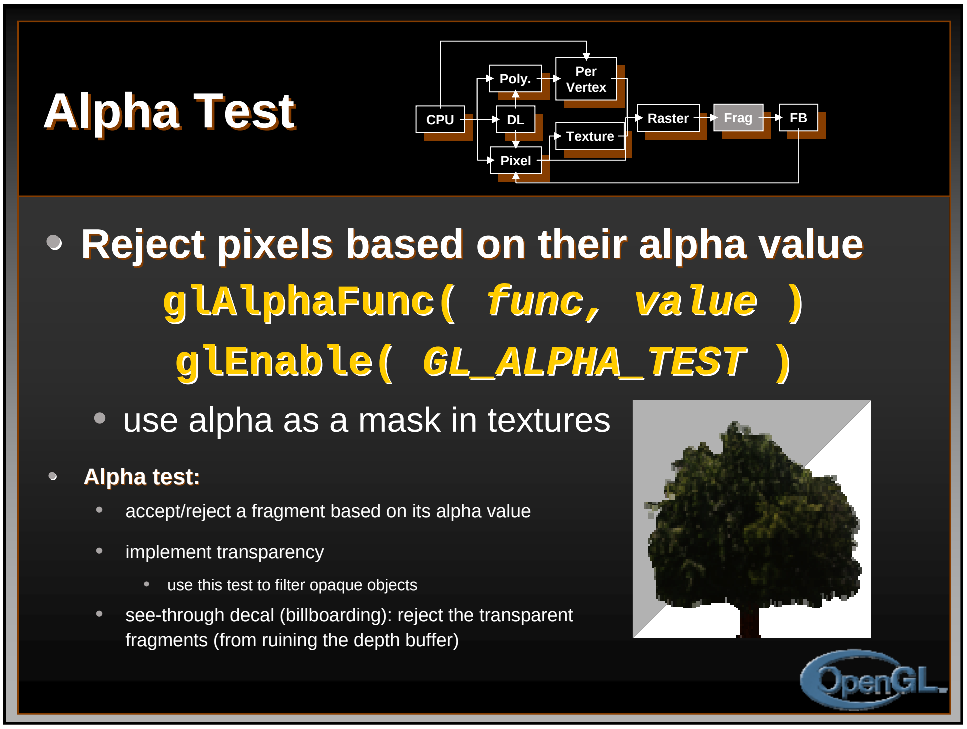
  - any fragments outside of box are clipped
  - useful for updating a small section of a viewport
    - affects **glClear()** operations



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();
}
```

OpenGL Scissor
OpenGL



Alpha Test

CPU
DL
Poly.
Per Vertex
Pixel
Texture
Raster
Frag
FB

- Reject pixels based on their alpha value

  glAlphaFunc(*func, value*)

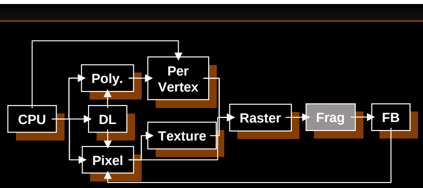
  glEnable(*GL_ALPHA_TEST*)

  - use alpha as a mask in textures

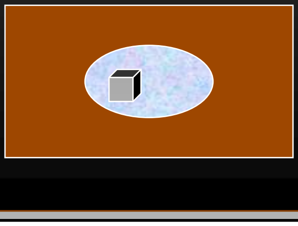
- **Alpha test:**
  - accept/reject a fragment based on its alpha value
  - implement transparency
    - use this test to filter opaque objects
  - see-through decal (billboarding): reject the transparent fragments (from ruining the depth buffer)

OpenGL

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

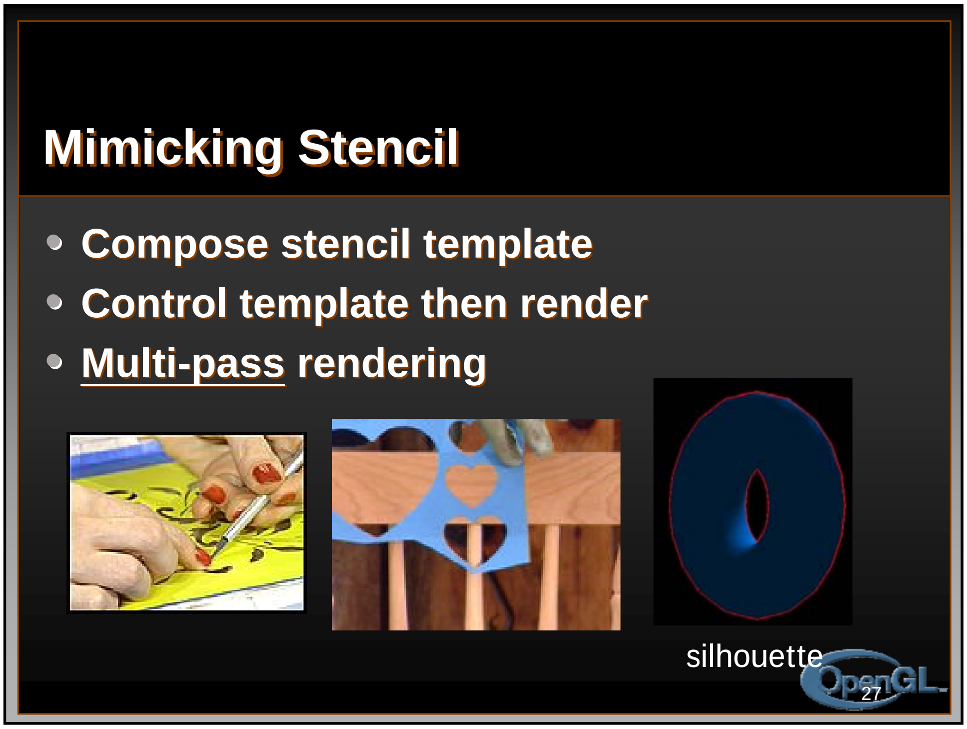


Stenciling

stencil

字義解釋 變化形 辨析

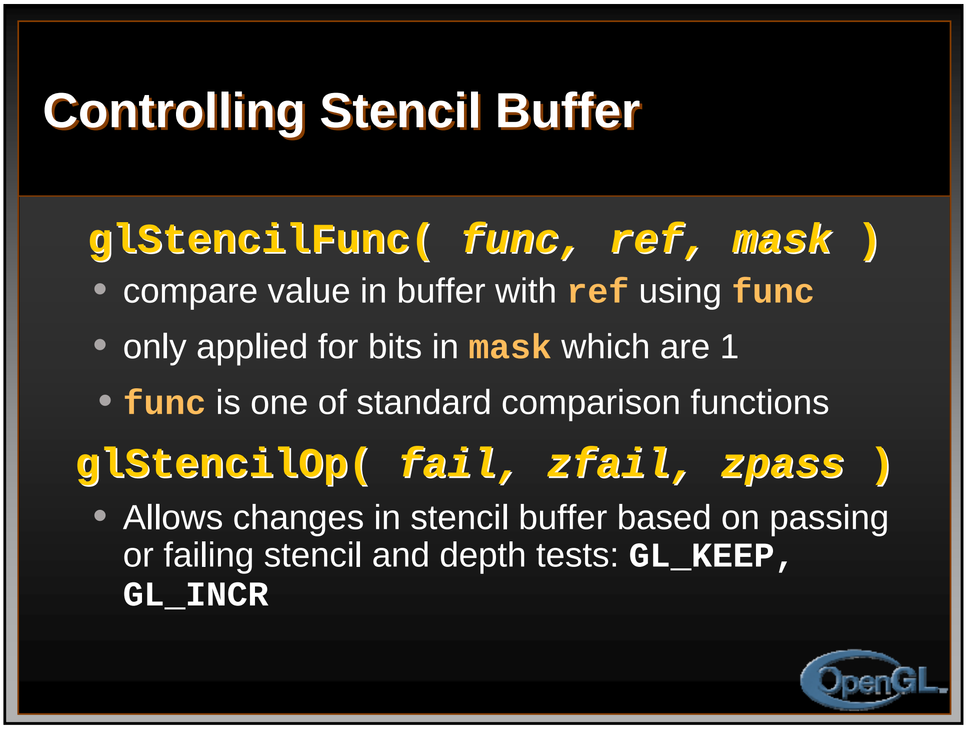
KK: [`stɛnsḷ]
DJ: [`stensl]
n.[C]
1. 印刷模板;油印蠟紙
2.(用模板或蠟紙印刷的)圖案,文字
vt.
1. 用模板印刷;用蠟紙印



Mimicking Stencil

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

silhouette



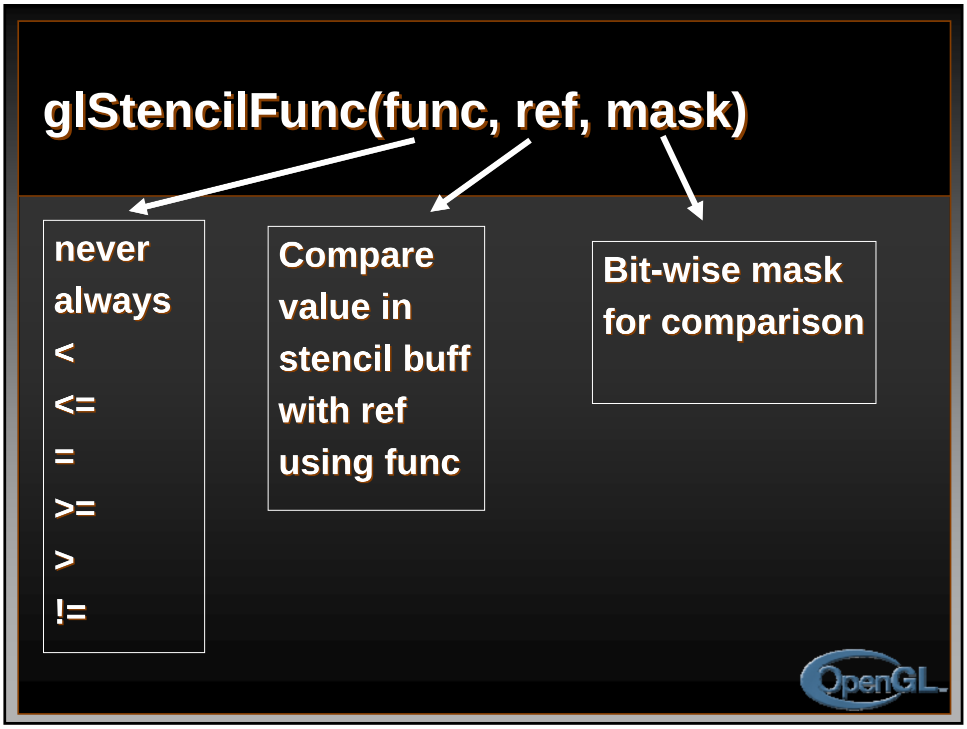
Controlling Stencil Buffer

`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`

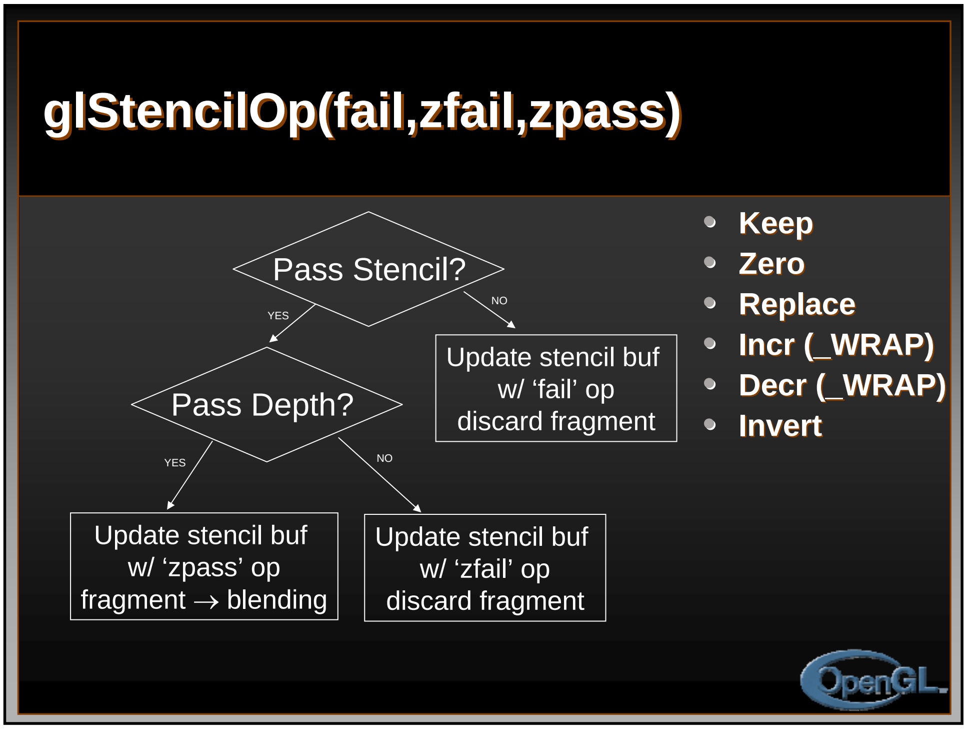


glStencilFunc(func, ref, mask)

never
always
<
<=
=
>=
>
!=

Compare value in stencil buff with ref using func

Bit-wise mask for comparison



glStencilOp(fail,zfail,zpass)

Pass Stencil?

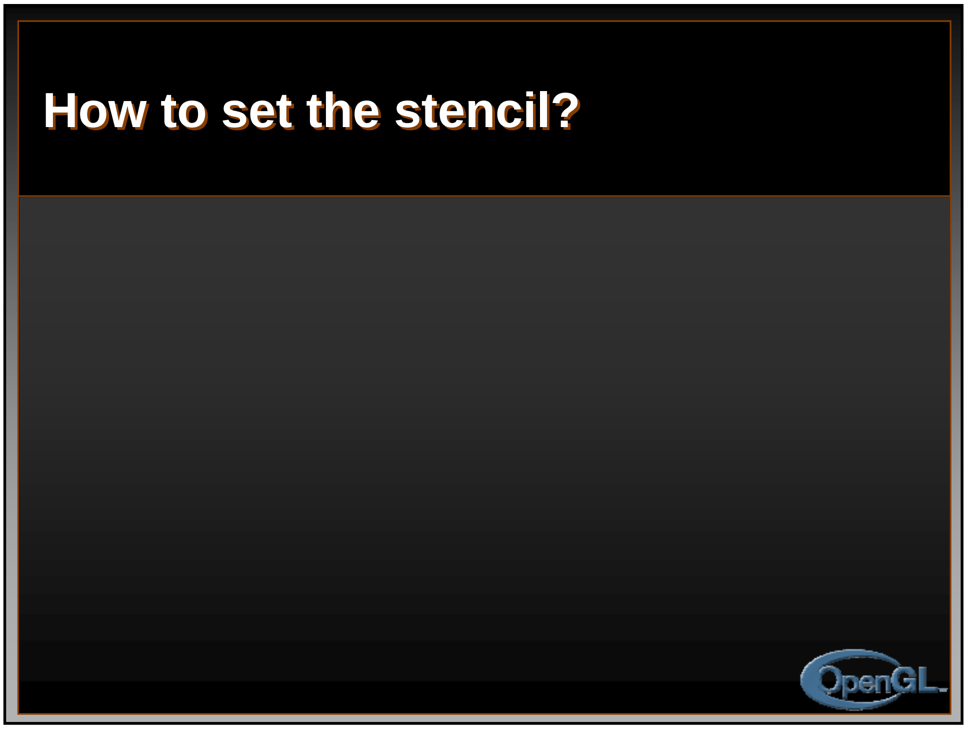
Update stencil buf w/ 'fail' op discard fragment

Pass Depth?

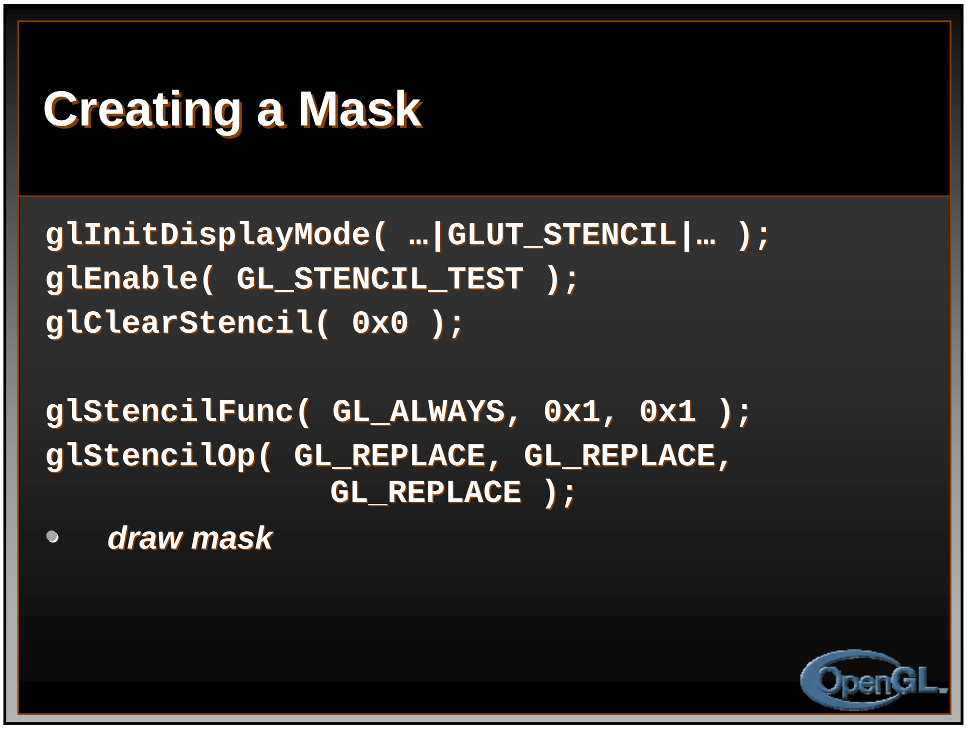
Update stencil buf w/ 'zpass' op fragment → blending

Update stencil buf w/ 'zfail' op discard fragment

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



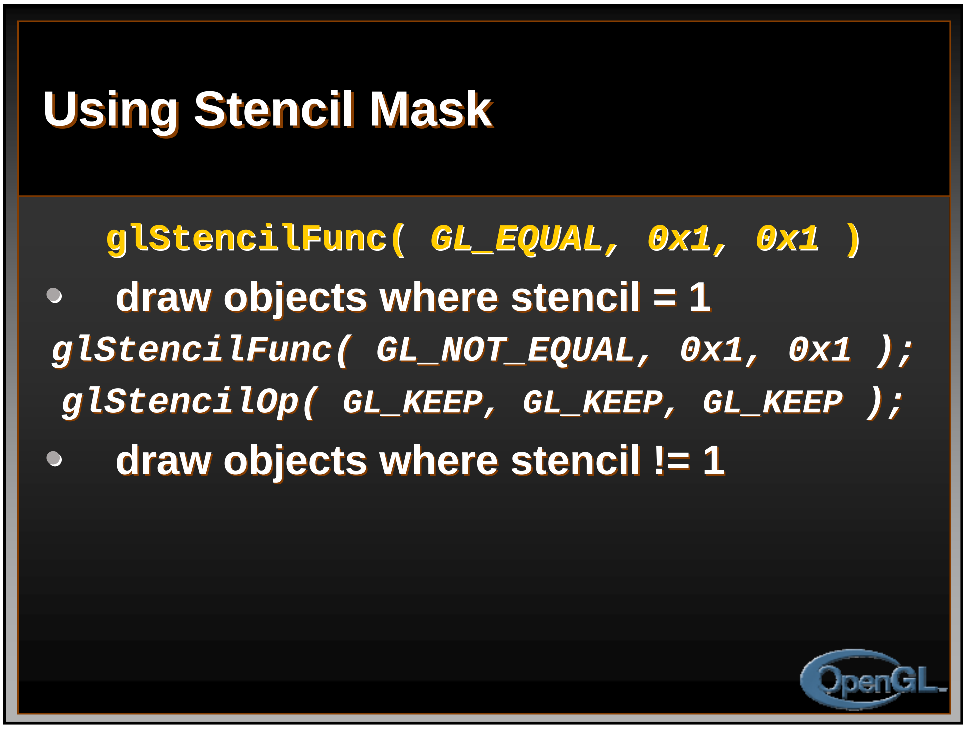
How to set the stencil?



Creating a Mask

```
glInitDisplayMode( …|GLUT_STENCIL|… );
glEnable( GL_STENCIL_TEST );
glClearStencil( 0x0 );

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

- *draw mask*



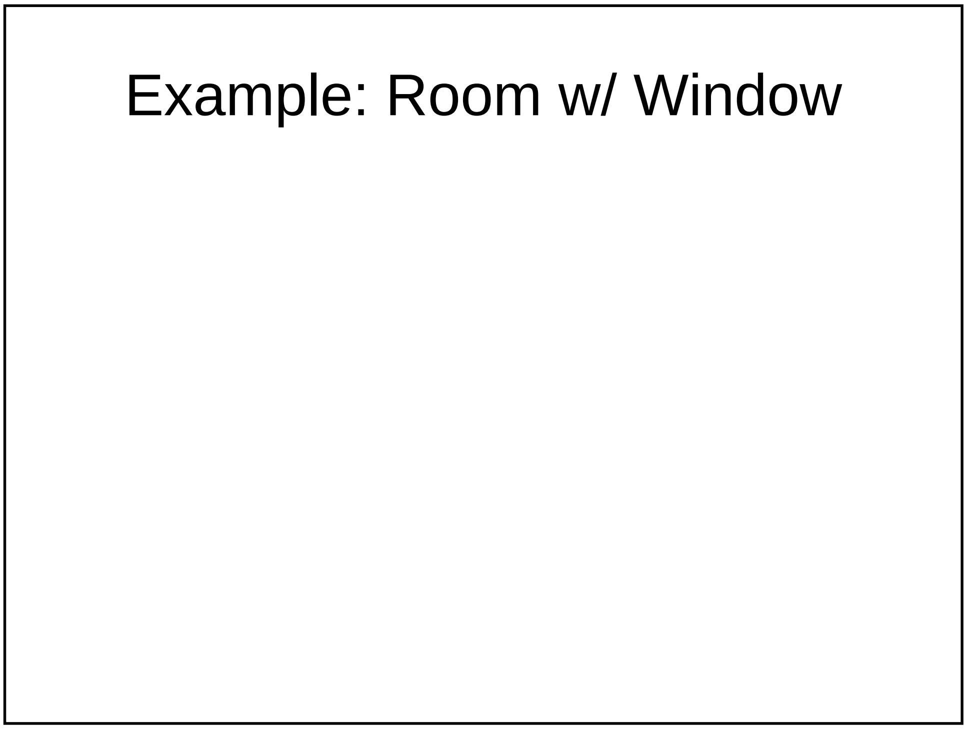
Using Stencil Mask

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

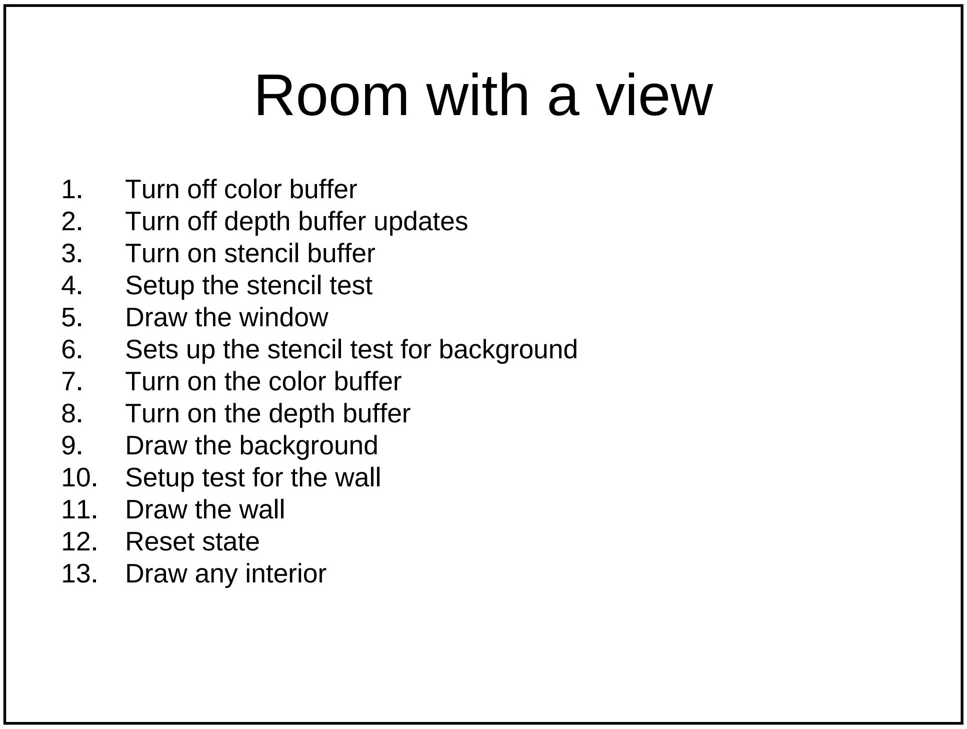
- draw objects where stencil = 1

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

- draw objects where stencil != 1

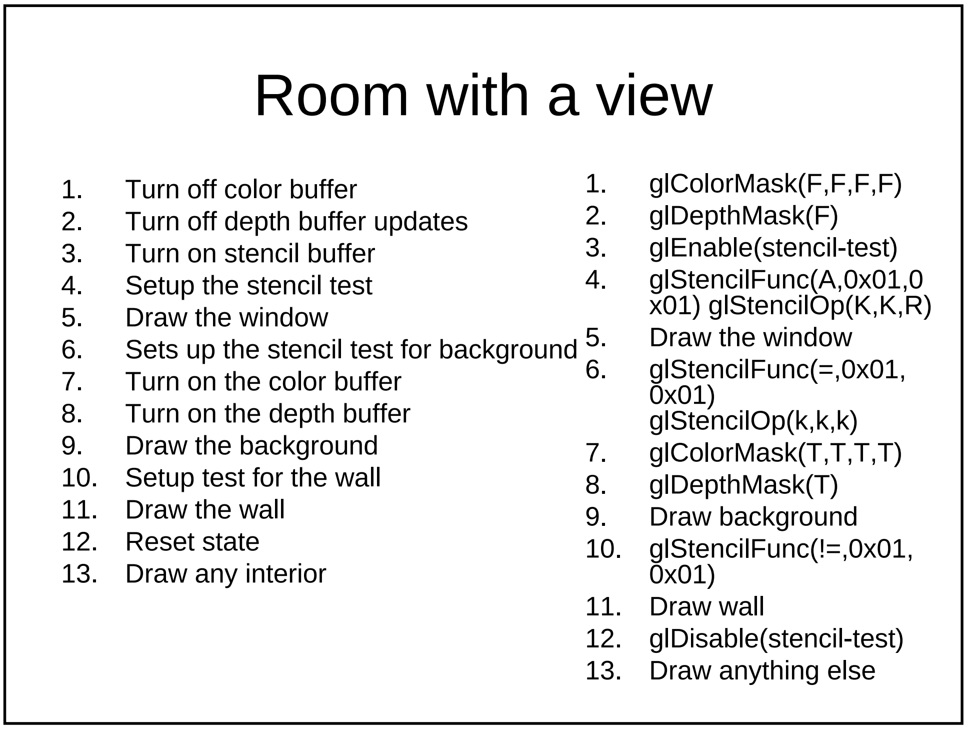


Example: Room w/ Window



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

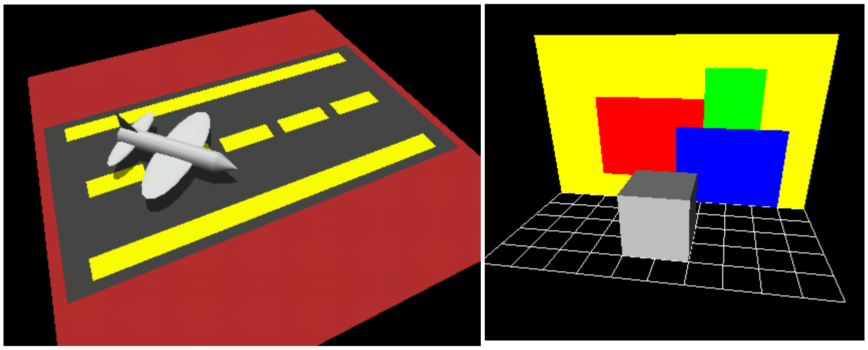


Room with a view

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

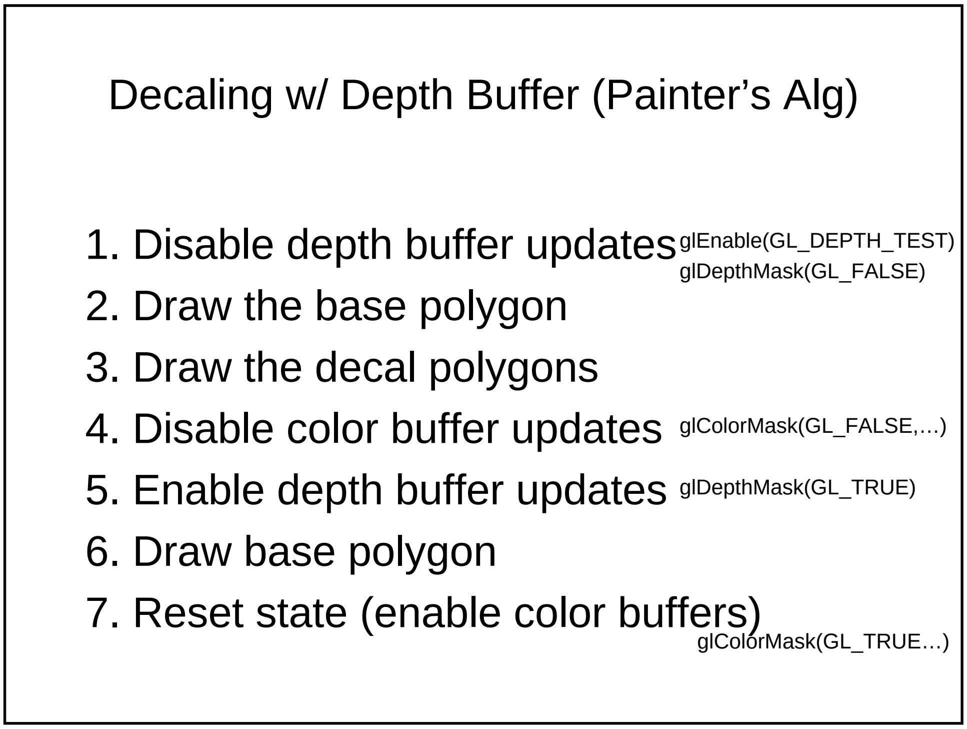


How to resolve z-fighting

37

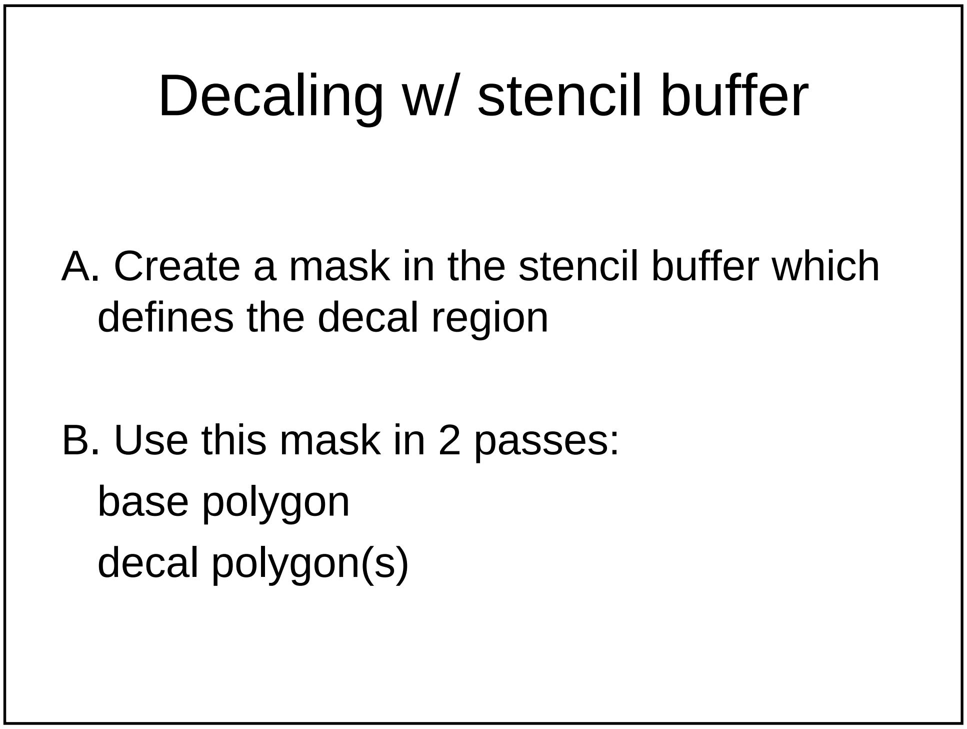
Decaling w/ Depth Buffer (Painter’s Alg)

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)

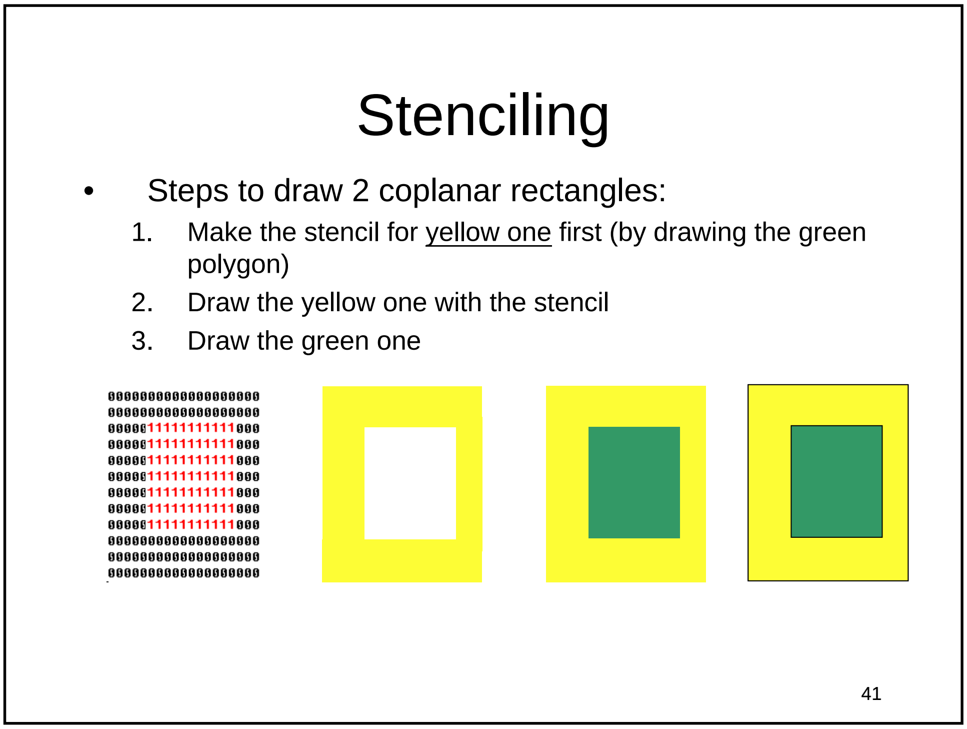
1. Disable depth buffer updates glEnable(GL_DEPTH_TEST)
   glDepthMask(GL_FALSE)
2. Draw the base polygon
3. Draw the decal polygons
4. Disable color buffer updates glColorMask(GL_FALSE,…)
5. Enable depth buffer updates glDepthMask(GL_TRUE)
6. Draw base polygon
7. Reset state (enable color buffers)
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

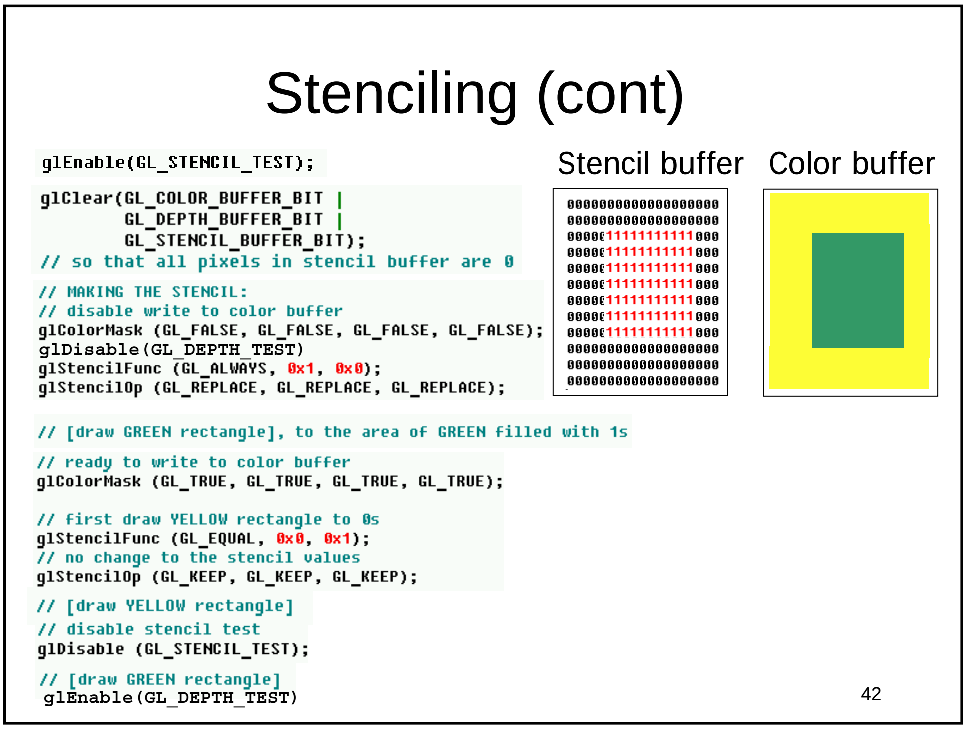


Stenciling

- 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

```
0000000000000000000
0000000000000000000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000000000000000000
0000000000000000000
0000000000000000000
```

41



Stenciling (cont)

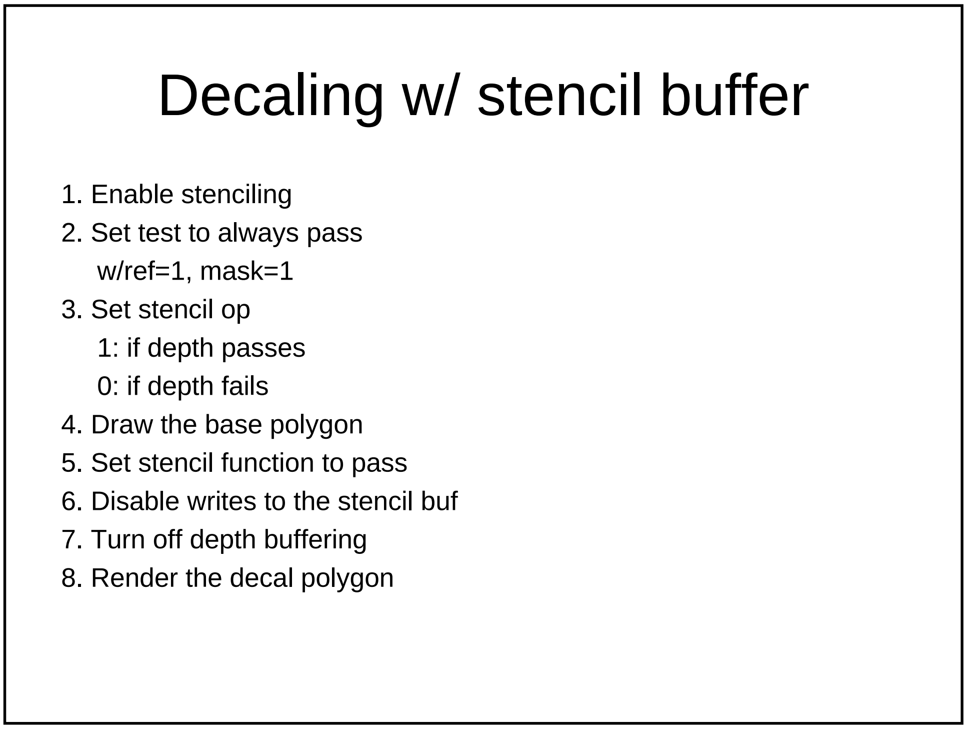
```
glEnable(GL_STENCIL_TEST);

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

```
0000000000000000000
0000000000000000000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000011111111111000
0000000000000000000
0000000000000000000
0000000000000000000
```

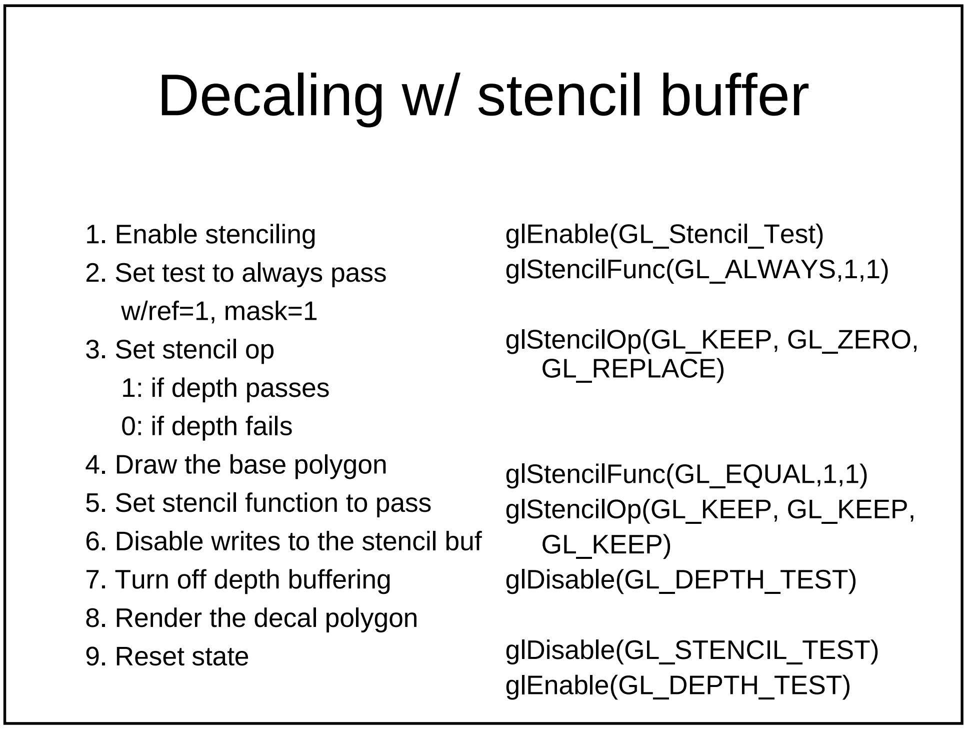
Color buffer

42



Decaling w/ stencil buffer

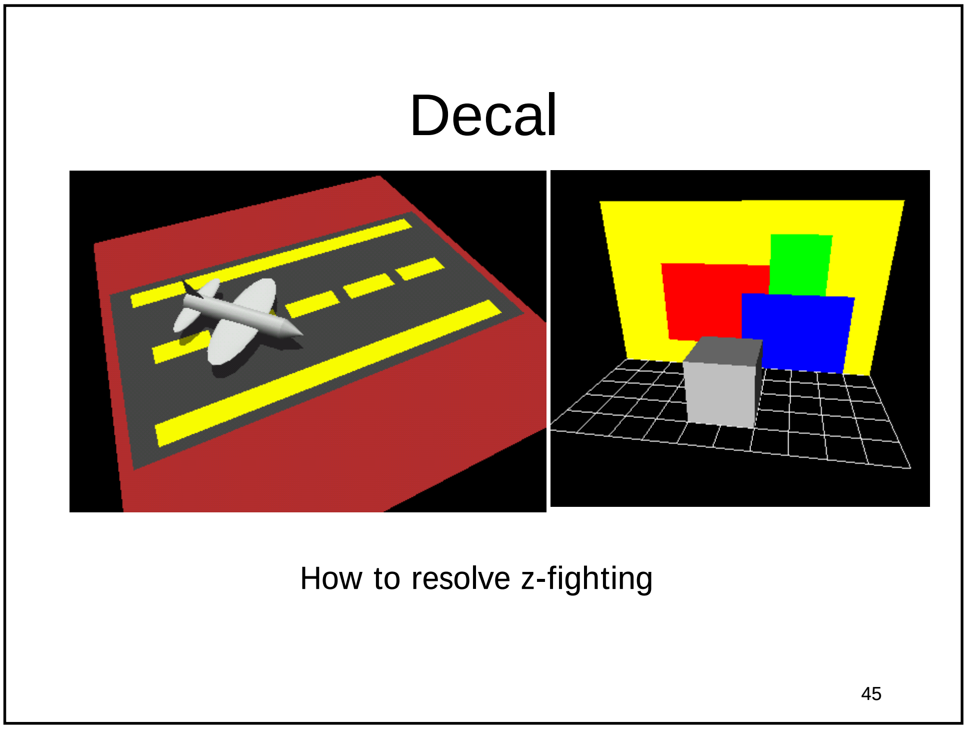
1. Enable stenciling
2. Set test to always pass
   w/ref=1, mask=1
3. Set stencil op
   1: if depth passes
   0: if depth fails
4. Draw the base polygon
5. Set stencil function to pass
6. Disable writes to the stencil buf
7. Turn off depth buffering
8. Render the decal polygon



Decaling w/ stencil buffer

1. Enable stenciling	glEnable(GL_Stencil_Test)
2. Set test to always pass	glStencilFunc(GL_ALWAYS,1,1)
w/ref=1, mask=1	
3. Set stencil op	glStencilOp(GL_KEEP, GL_ZERO, GL_REPLACE)
1: if depth passes	
0: if depth fails	
4. Draw the base polygon	
5. Set stencil function to pass	glStencilFunc(GL_EQUAL,1,1)
6. Disable writes to the stencil buf	glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP)
7. Turn off depth buffering	glDisable(GL_DEPTH_TEST)
8. Render the decal polygon	

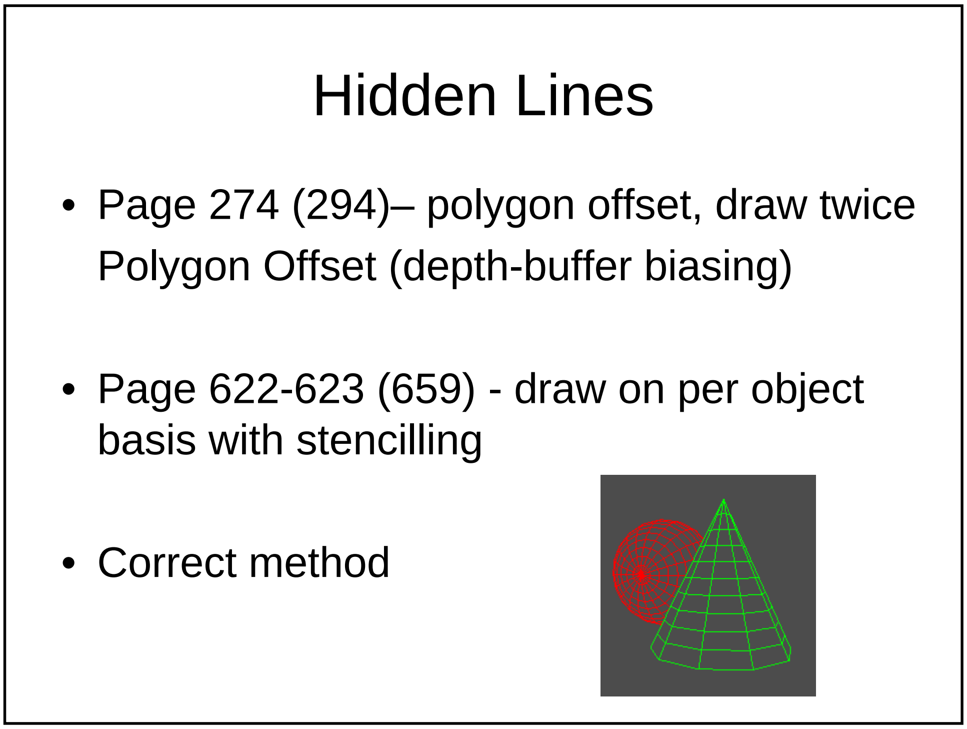
9. Reset state	glDisable(GL_STENCIL_TEST)
	glEnable(GL_DEPTH_TEST)



Decal

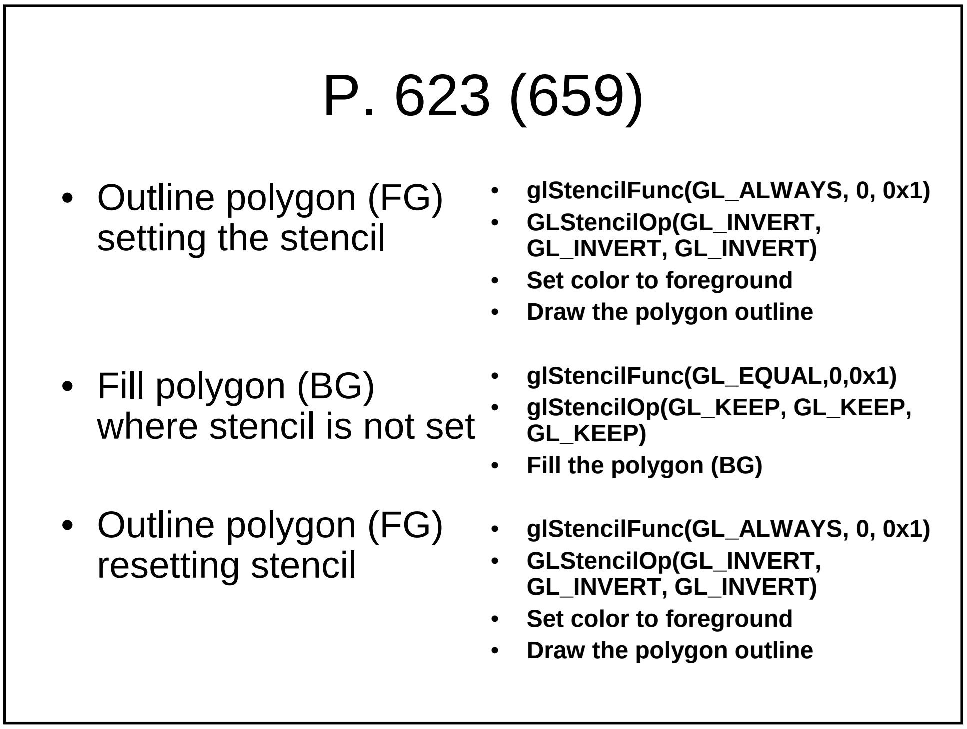
How to resolve z-fighting

45



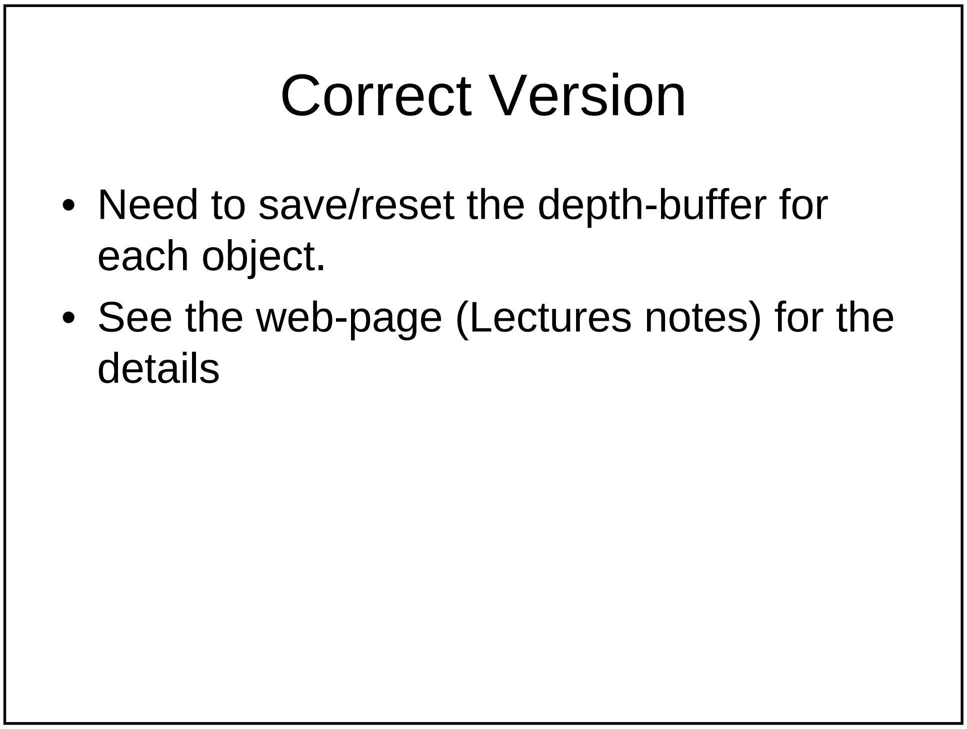
Hidden Lines

- Page 274 (294)– polygon offset, draw twice
  Polygon Offset (depth-buffer biasing)
- Page 622-623 (659) - draw on per object basis with stencilling
- Correct method



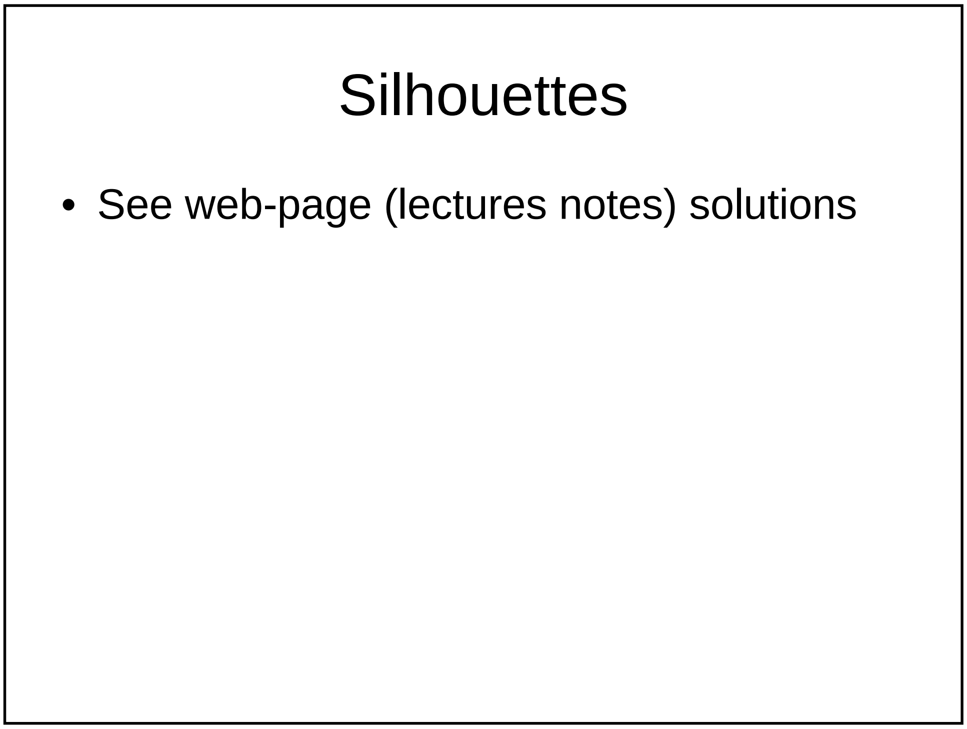
P. 623 (659)

- 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



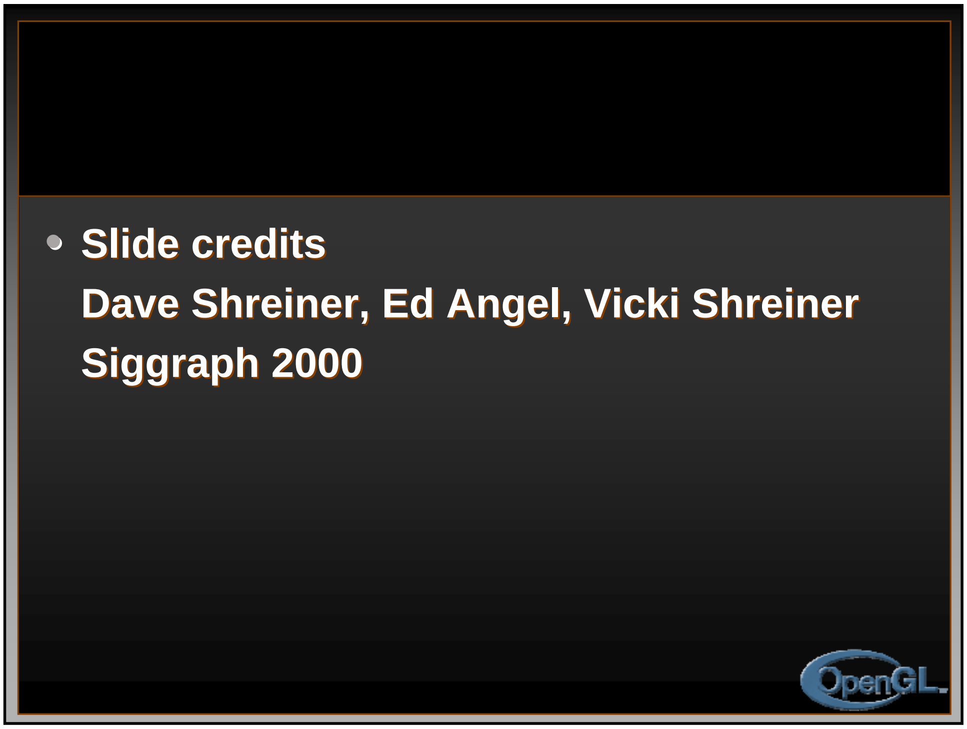
Correct Version

- 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