Using the Stencil Buffer

The Framebuffers

Here's what the Stencil Buffer can do for you:

1. While drawing into the Back Buffer, you can write values into the Stencil Buffer at the same time.
2. While drawing into the Back Buffer, you can do arithmetic on values in the Stencil Buffer at the same time.
3. The Stencil Buffer can be used to write-protect certain parts of the Back Buffer.
You Can Think of the Stencil Buffer as a Separate Framebuffer, or, You Can Think of it as being Per-Pixel

Both are correct, but I like thinking of it "per-pixel" better.

The Stencil Buffer is Tested Per-Pixel, Very Much Like the Depth Buffer is

```c
(glStencilFunc( func, ref, mask )

This specifies the comparison test that is to be done per-pixel.

`func` can be any of GL_NEVER, GL_ALWAYS, GL_EQUAL, GL_NOTEQUAL, GL_LESS, GL_LEQUAL, GL_GREATER, GL_GEQUAL

`ref` is an integer reference value that is used to test the pixel's existing stencil value against using the chosen `func`

`mask` is set to 1 in all these examples

The stencil test produces a true or false value at each pixel where drawing is to be done.

```
if( ref <func> S_{existing} is true )
{
    Allow the color write to the existing pixel to take place;
    Modify the pixel's existing stencil value depending on what the glStencilOp says to do;
}
```
glStencilOp( sfail, zfail, zpass )

This specifies how a pixel's stencil value is modified when a fragment passes or fails the stencil test depending on what combinations of true and false the stencil test and the depth buffer test produce. If the stencil test fails, then sfail happens. If the stencil test succeeds, then either zfail or zpass happen depending on if the depth-buffer test failed or succeeded.

The three values can be any of:
- GL_KEEP: Retain the existing stencil value
- GL_ZERO: Set the stencil value to zero
- GL_REPLACE: Replace the stencil value with ref from the Stencil Func
- GL_INCR: Increment the stencil value, with clamping
- GL_INCR_WRAP: Increment the stencil value, without clamping
- GL_DECR: Decrement the stencil value, with clamping
- GL_DECR_WRAP: Decrement the stencil value, without clamping
- GL_INVERT: Bitwise toggle the stencil bits: 0's → 1's, 1's → 0's

This Tells You What to Do with the true or false Value from the Stencil Test

if (ref < func) Sexisting is true
{
    Allow the color write to the existing pixel to take place;
    Modify the pixel's existing stencil value depending on what the glStencilOp says to do;
}

Setting Up the Stencil Buffer

// at the top of the program:
const int STENCILBIT = 1;
const int DEFAULT_STENCIL = 0;
const float BIGX = 2.;
const float BIGY = BIGX;
const float CLOSEZ = -1.;
float Xlens, Ylens;
float Box = 0.40f;

// in InitGraphics( ):
glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH | GLUT_STENCIL );
glClearColor( BACKGROUND_COLOR );
glClearStencil( DEFAULT_STENCIL );

// in Display( ):
... glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
... glEnable( GL_STENCIL_TEST );
...
Using the Stencil Buffer to Create a Magic Lens

1. Clear the SB = 0
2. Enable the SB
3. Write protect the color buffer and depth buffer
4. Draw a filled square, while setting SB = 1
5. Write-enable the color buffer and depth buffer
6. Draw the solids wherever SB == 0
7. Draw the wireframes wherever SB == 1
8. Disable the SB
Moving the Magic Lens with the Middle Mouse Button

```c
// in MouseMotion( ):
if( ActiveButton & MIDDLE )
{
  if( Stencil == LENS )
  {
    int w = glutGet( GLUT_WINDOW_WIDTH );
    int h = glutGet( GLUT_WINDOW_HEIGHT );
    Xlens = 2.*(float)x/(float)w - 1.;
    Ylens = -2.*(float)y/(float)h + 1.;
  }
  else
  {
    Scale += SCLFACT * (float) ( dx - dy );
  }
}
```

Moving the Magic Lens with the Middle Mouse Button

```c
x/w ranges from 0. to 1.
y/h ranges from 1. to 0
Xlens and Ylens range from -1. to 1. (NDC)
```

Using the Stencil Buffer to Create a Magic Lens

```c
glMatrixMode( GL_PROJECTION );
glLoadIdentity();

glMatrixMode( GL_MODELVIEW );
glLoadIdentity();

These two identity transformation matrices cause
the drawing to take place in NDC (-1 to 1.), which
is what Xlens, Ylens, and Box are defined in

```c
glDepthMask( GL_FALSE );
glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );

```c
glStencilFunc( GL_ALWAYS, 1, STENCILBIT );
glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );

```c
Everywhere we draw, always
replace the stencil value with a 1
```

```c
gBegin( GL_QUADS );
gVertex2f( Xlens-Box/2., Ylens-Box/2. );
gVertex2f( Xlens+Box/2., Ylens-Box/2. );
gVertex2f( Xlens+Box/2., Ylens+Box/2. );
gVertex2f( Xlens-Box/2., Ylens+Box/2. );
gEnd( );
```

```c
Draw a filled-in box
```

```c
gColorMask( GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE );
gDepthMask( GL_TRUE );
```

```c
Write-enable the depth and color buffers
```
Using the Stencil Buffer to Create a Magic Lens

```c
<< set the GL_PROJECTION and GL_MODELVIEW matrices as normal >>

glEnable( GL_LIGHTING );
glStencilFunc( GL_EQUAL, 0, STENCILBIT );
glShadeModel( GL_SMOOTH );
for( int i = 0; i < 8; ++i )
{
    glCallList( SolidLists[i] );
}
glDisable( GL_LIGHTING );

<< set the GL_PROJECTION and GL_MODELVIEW matrices to identity again >>

gDisable( GL_DEPTH_TEST );
gColor3f( 1., 1., 1. );
gBegin( GL_LINE_LOOP );
gVertex2f( Xlens-Box/2., Ylens-Box/2. );
gVertex2f( Xlens+Box/2., Ylens-Box/2. );
gVertex2f( Xlens+Box/2., Ylens+Box/2. );
gVertex2f( Xlens-Box/2., Ylens+Box/2. );
gEnd();
gEnable( GL_DEPTH_TEST );
```

I Once Used the Stencil Buffer to Create a Magic Lens for Volume Data

In this case, the scene inside the lens was created by drawing the same object, but drawing it with its near clipping plane being farther away from the eye position.
Using the Stencil Buffer to Perform Polygon Capping

1. Clear the SB = 0
2. Enable the SB
3. Draw the polygons, setting SB = ~ SB: 0's → 1's, 1's → 0's
4. Draw a large gray polygon in front of the entire scene wherever SB != 0
5. Disable the SB
As we draw the **solid** objects, always invert the stencil bits:
0's → 1's
1's → 0's

Because these were all **solid** objects, they had a front face and a back face drawn. Thus, most of the time, the SB values got inverted back to 0. If they didn’t, that means that the solid object penetrated the near clipping plane and now needs to be capped.

```c
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );

glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );

glDisable( GL_LIGHTING );
glDisable( GL_LIGHT0 );
glStencilFunc( GL_NOTEQUAL, 0, STENCILBIT );
glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );
glShadeModel( GL_FLAT );
glColor3f( .5f, .5f, .5f );
gBegin( GL_QUADS );
glVertex3f( -BIGX, -BIGY, CLOSEZ );
glVertex3f(  BIGX, -BIGY, CLOSEZ );
glVertex3f(  BIGX,  BIGY, CLOSEZ );
glVertex3f( -BIGX,  BIGY, CLOSEZ );
gEnd( );
```

**Using the Stencil Buffer to Perform Polygon Capping**

Using the Stencil Buffer to Better Outline Polygons

**Before**

**After**

Z-fighting
Using the Stencil Buffer to Better Outline Polygons

Clear the SB = 0
Enable the SB
for( each polygon )
{
    Draw the edges, setting SB = 1
    Draw the filled polygon wherever SB  !=  1
    Draw the edges again, setting SB = 0
}
Disable the SB

Before
After

Outlining Polygons the Naïve Way

1. Draw all polygons
2. Draw all edges

Z-fighting
Using the Stencil Buffer to Better Outline Polygons

for( int f = 0; f < NumFaces; f++ )
{
    glStencilFunc( GL_ALWAYS, 1, STENCILBIT );
    glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );
    glShadeModel( GL_FLAT );
    glColor3f( 1., 1., 1. );
    glBegin( GL_LINE_LOOP );
    for( int v = FirstVertex[f];  v < FirstVertex[f+1];  v++ )
    {
        glVertex3f( Vertices[v].x, Vertices[v].y, Vertices[v].z );
    }
    glEnd( );
    glStencilFunc( GL_EQUAL, 0, STENCILBIT );
    glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );
    glEnable( GL_LIGHTING );
    glShadeModel( GL_SMOOTH );
    glMaterialfv( ... );
    glBegin( GL_POLYGON );
    for( int v = FirstVertex[f];  v < FirstVertex[f+1];  v++ )
    {
        glNormal3f( Normals[v].x, Normals[v].y, Normals[v].z );
        glVertex3f( Vertices[v].x, Vertices[v].y, Vertices[v].z );
    }
    glEnd( );
    glStencilFunc( GL_ALWAYS, 0, STENCILBIT );
    glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );
    glShadeModel( GL_FLAT );
    glColor3f( 1., 1., 1. );
    glBegin( GL_LINE_LOOP );
    for( int v = FirstVertex[f];  v < FirstVertex[f+1];  v++ )
    {
        glVertex3f( Vertices[v].x, Vertices[v].y, Vertices[v].z );
    }
    glEnd( );

Put "masking tape" down on the polygon edges

Paint the polygon, which also paints the edges

Pull the "masking tape" up and paint just the polygon edges
Using the Stencil Buffer to Perform Hidden Line Removal

Clear the SB = 0
Enable the SB

for( each polygon )
{
    Draw the edges, setting SB = 1
    Draw the polygon, unlit and flat shaded, in the background color wherever SB != 1
    Draw the edges again, setting SB = 0
}

Disable the SB
Hidden Line Removal in Pre-Vis for the 2019 Movie *Dumbo*