Introduction

The illusion of realism can be increased by adding reflecting objects that lie above reflective surfaces like water or ice. OpenGL offers a very simple way to generate reflections using a negative scale and alpha blending. The reflections can be automatically restricted to the on-screen extent of the reflective surface using the stencil buffer.
Reflection via Negative Scale

Although there are many ways to generate reflections, OpenGL (and other graphics APIs) offer a very simple way to create this effect:

1. Assume the reflective surface is centered about the origin and lies in the XZ-plane, and that the object(s) to be reflected lie in the +Y-halfplane.
2. Draw the object(s) above the reflective surface, exactly as normal.
3. "Reflect" the object(s) about the reflective surface via glScalef( 1, -1, 1 ); this "flips" the Y-axis through the reflective surface.
4. Draw the object(s) again, exactly as before; this time they will appear reflected below the reflective surface.
5. Enable alpha-blending via glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA ).
6. Draw the reflective surface with some amount of transparency; this will allow you to "see through" to the reflected objects that lie beneath the surface.

The results is an image that looks something like this:
One major problem exists with the previous implementation, however. If the reflective surface doesn't entirely cover the reflected version(s) of the object(s), the will appear beneath the surface and uncover the effect:

![Image of a lizard reflected in a plane]

Notice here the viewer can see that there's an "extra" lizard underneath the reflecting plane, because it's seen outside the extents of the plane.

What we need is to only draw the reflected object(s) in areas of the screen that also contain the reflected surface. OpenGL offers a very efficient way to do this via the stencil buffer.

The stencil buffer is an auxiliary buffer that stores information as parts of the scene are rendered. Normally, the number of stencil entries is identical to the number of framebuffer and Z-buffer entries, that is, one stencil entry is
available for every pixel rendered in a scene. Stenciling is typically used in multipass rendering algorithms to mask out portions of the screen for decals, outlining, and constructive solid geometry rendering.

The `glStencilFunc(_func, _ref, _mask)` routine defines a test function _func_, a reference value _ref_, and a bitmask _mask_. The stencil function performs a test of the stencil value at the pixel position being tested anded with the bitmask against the reference value anded with the bitmask. If the test passes, the pixel is allowed to continue. If the test fails, the pixel is eliminated from further processing.

The following test functions are available, with associated meanings:

- GL_NEVER, always fails
- GL_LESS, pass if ( ref & mask ) < ( stencil & mask )
- GL_LEQUAL, pass if ( ref & mask ) ≤ ( stencil & mask )
- GL_GREATER, pass if ( ref & mask ) > ( stencil & mask )
- GL_GEQUAL, pass if ( ref & mask ) ≥ ( stencil & mask )
- GL_EQUAL, pass if ( ref & mask ) = ( stencil & mask )
- GL_NOTEQUAL, pass if ( ref & mask ) ≠ ( stencil & mask )
- GL_ALWAYS, always pass

The stencil buffer can be initialized with:

```c
glClearStencil( 0 );
glClear( GL_STENCIL_BUFFER_BIT );
```

The default values for _func_, _ref_, and _mask_ are GL_ALWAYS, 0, and 1, respectively.

The stencil buffer values are updated based on whether the stencil test and the depth test (if it's active) pass or fail. The `glStencilOp( fail, zfail, zpass )` defines three update actions to take when:

- fail for when the stencil test fails
- **zfail** for when the stencil test passes, but the depth test fails
- **zpass** for when both the stencil test and the depth test pass

Possible values for the update actions are:

- **GL_KEEP**, keep the current value
- **GL_ZERO**, set stencil value to 0
- **GL_REPLACE**, set stencil value to ref
- **GL_INCR**, increment stencil value
- **GL_DECR**, decrement stencil value
- **GL_INVERT**, bitwise invert stencil value
Reflection Masking via Stenciling

How can we use the stencil buffer to restrict the reflected object to the reflective surface? We can "stencil" the location of the reflective surface, then only allow pixels in the reflected object to be rendered within this location.

First, we render the reflective surface to initialize the stencil values to 1 wherever the surface is rendered:

```c
// Don't update either colour or depth values

glDisable( GL_DEPTH_TEST );
glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );

// Draw reflective surface, tag positions with stencil value 1

glEnable( GL_STENCIL_TEST );
glStencilFunc( GL_ALWAYS, 1, 0xffffffff );
glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );
draw_reflective_surface();

The above code always (GL_ALWAYS) replaces (GL_REPLACE) the stencil value with reference value 1 in every position where a pixel for the reflective surface occurs.
```

Once this is done, when we draw the reflected object(s), we only allow...
drawing to occur in positions with a stencil value of 1:

```c
    glEnable( GL_STENCIL_TEST );
    glStencilFunc( GL_EQUAL, 1, 0xffffffff );
    glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );

    glPushMatrix();
    glScalef( 1, -1, 1 );
    draw_objects();
    glPopMatrix();
    glDisable( GL_STENCIL_TEST );
```

This above code fragment only allows a pixel to pass the stencil test if the stencil value equals (GL_EQUAL) reference value 1, that is, when ( 1 & 0xffffffff ) = ( stencil & 0xffffffff ). Regardless of whether the tests pass or fail, we keep (GL_KEEP) the current stencil value.

The resulting image is exactly what we wanted: