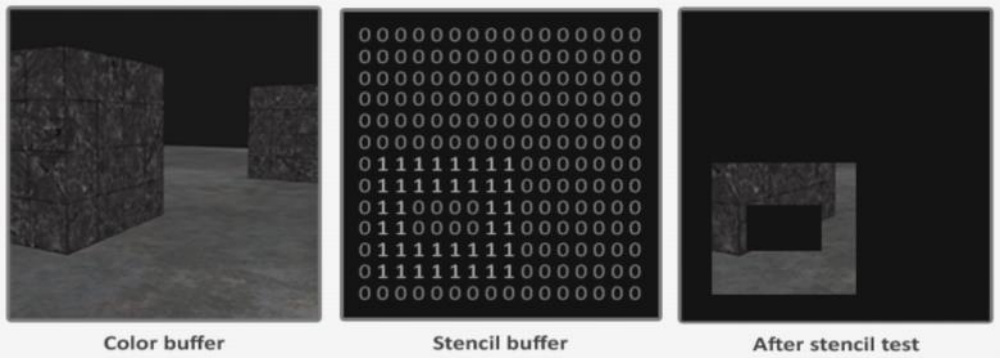
STENCIL TESTING

Once the fragment shader has processed a fragment, a stencil test is executed that, just like the depth test, has the option to discard fragments. After that, the remaining fragments are passed to the depth test for OpenGL to also possibly discard.

The stencil test is based on the content of the stencil buffer, which can be updated during rendering.

- The stencil buffer usually contains 8 bits per stencil value, which amounts to 256 different stencil values per pixel. We can set these stencil values to values of our liking and we can discard or keep fragments whenever a particular fragment has a certain stencil value.
  - The stencil buffer is sometimes set up by the windowing library (GLFW does do this), so we don't need to create one, but some windowing libraries do not.
- The image to the right demonstrates an idea of how the stencil buffer considers culling fragments, though the pixels are not to scale.



We can use stencil buffer operations to write specific values to the stencil buffer, then read these values to discard or pass certain fragments.

The general outline of using the stencil buffer is as follows:

1. Enable writing to the stencil buffer
2. Render objects, updating the content of the stencil buffer
3. Disable writing to the stencil buffer
4. Render (other) objects, this time discarding certain fragments based on the content of the stencil buffer.

By using the stencil buffer, we can thus discard certain fragments based on the fragments of other drawn objects in the scene.

You can enable stencil testing with `glEnable(GL_STENCIL_TEST)`. Also, don't forget to clear the stencil buffer bit before each frame using `glClear(GL_STENCIL_BUFFER_BIT)`.

Just like the depth buffer's `glDepthMask` function, the stencil buffer has an equivalent `glStencilMask` function that allows us to set a **bitmask** that is AND'd with the stencil value about to be written to the buffer. Meaning, in a bitmask, a 1 bit can be written to and 0 bit cannot.

- `glStencilMask(0xFF)` is the default for the stencil mask, which writes each bit to the stencil buffer as is.
  - `0xFF` is `11111111` in binary, which is why all 8 bits in the bitmask can be written to.
- `glStencilMask(0x00)` writes each bit as a 0 to the stencil buffer, which disables writing to the stencil buffer entirely.
  - `0x00` is `00000000` in binary, which is why none of the 8 bits in the bitmask can be written to.

Stencil Functions

Like with depth testing, we have a certain amount of control over when a stencil test should pass or fail and how it should affect the stencil buffer.

`glStencilFunc(GLenum func, GLint ref, GLuint mask)` has 3 parameters:

1. **`func`:** Sets the stencil function that determines whether a fragment passes or is discarded. This test function is applied to the stored stencil value and the `glStencilFunc`'s ref value.
   - It has the same possible options as glDepthFunc, which are `GL_NEVER`, `GL_LESS`, `GL_LEQUAL`, `GL_GREATER`, `GL_GEQUAL`, `GL_EQUAL`, `GL_NOTEQUAL` and `GL_ALWAYS`.
2. **`ref`:** Specifies the reference value for the stencil test. The stencil buffer's content is compared to this value.
3. **`mask`:** Specifies a mask that is AND'd with both the reference value and the stored stencil value before the test compares them. Initially set to all 1s.

Example: `glStencilFunc(GL_EQUAL, 1, 0xFF)` tells OpenGL that whenever the stencil value of a fragment is equal to the reference value 1, the fragment passes the test and is drawn. Otherwise, the fragment is discarded.

`glStencilFunc` describes whether OpenGL should pass or discard fragments based on the stencil buffer's content, but `glStencilOp` allows us to update the stencil buffer.
`glStencilOp(GLenum sfail, GLenum dpfail, GLenum dppass)` has 3 parameters:

1. **`sfail`**: Action to take if the stencil test fails.
2. **`dpfail`**: Action to take if the depth test fails.
3. **`dppass`**: Action to take if both the stencil and depth test pass.

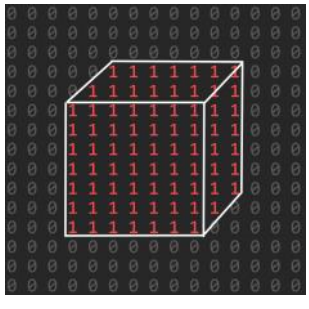
You can pass any of the arguments in the image on the right to each parameter.

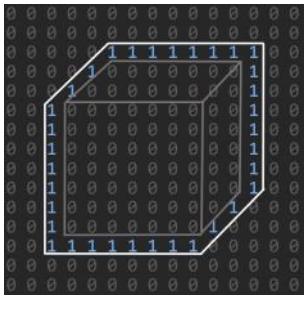
By default `glStencilOp` is set to `GL_KEEP` for all 3 parameters, so the stencil buffer keeps its values, no matter the outcome of the tests.

Action	Description
GL_KEEP	The currently stored stencil value is kept.
GL_ZERO	The stencil value is set to 0.
GL_REPLACE	The stencil value is replaced with the reference value set with glStencilFunc.
GL_INCR	The stencil value is increased by 1 if it is lower than the maximum value.
GL_INCR_WRAP	Same as GL_INCR, but wraps it back to 0 as soon as the maximum value is exceeded.
GL_DECR	The stencil value is decreased by 1 if it is higher than the minimum value.
GL_DECR_WRAP	Same as GL_DECR, but wraps it to the maximum value if it ends up lower than 0.
GL_INVERT	Bitwise inverts the current stencil buffer value.

Object Outlining

With object outlining can be achieved with just the stencil buffer. For each object, we create a small colored border around the (combined) objects.

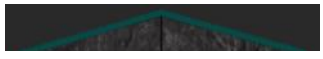




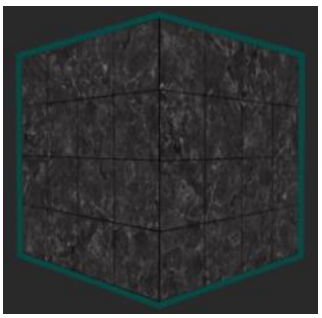
The routine for outlining objects is as follows:

1. Enable stencil testing.
2. Set the stencil op to GL_ALWAYS before drawing the (to-be-outlined) objects, updating the stencil buffer with 1s wherever the objects' fragments are rendered.
3. Render the objects.
4. Disable stencil writing and depth testing.
5. Scale each of the objects by a small amount.
6. Use a different fragment shader that outputs a single (border) color.
7. Draw the objects again, but only if their fragments' stencil values are not equal to 1.
8. Enable depth testing again and restore stencil func to GL_ALWAYS.

This process sets the content of the stencil buffer to 1s for each of the object's fragments. When it is time to draw the borders,



7. Draw the objects again, but only if their fragments' stencil values are not equal to 1.
8. Enable depth testing again and restore stencil func to GL_ALWAYS.



This process sets the content of the stencil buffer to 1s for each of the object's fragments. When it is time to draw the borders, we draw scaled-up versions of the objects only where the stencil test passes; we discard all the fragments of the scaled-up versions that are part of the original objects' fragments using the stencil buffer.

Implement object outlining for objects in your scene.

- Object Outlining using Stencil Buffer
- **NOTE:** With models that are not geometrically symmetrical (like a cube), the above highlighting methodology will not yield the results you likely seek. A better, more verbose outlining system would work by scaling the vertices of the outline model outwardly along their normal direction vectors.
  - Better Object Outlining

Stencil testing is used for more purposes besides outlining objects, such as drawing textures inside a rear-view mirror so it nearly fits into the mirror shape, or rendering real-time shadows with a stencil buffer technique called **shadow volumes**.

Object Outlining using Stencil Buffer

Thursday, April 21, 2022 5:37 PM

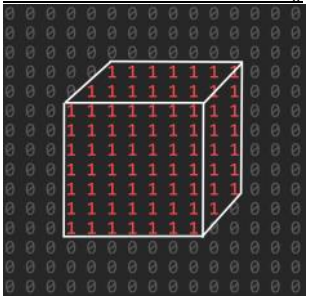
outline.frag

```
#version 330 core

out vec4 FragColor;

uniform vec3 outlineColor;

void main() {
   FragColor = vec4(outlineColor, 1.0);
}
```

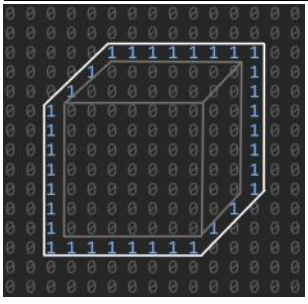
Render Loop

```
glEnable(GL_DEPTH_TEST);
// If any of the tests fail, we do nothing. If both tests pass, we replace the stored stencil value with the
// reference value of glStencilFunc.
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT); // Clear the stencil buffer bit before each frame is rendered

glStencilMask(0x00); // Make sure not to update the stencil buffer while drawing objects we don't want to outline
objectShader.use();
DrawFloor()

// Draws the to-be-outlined objects and updates the stencil buffer with a 1 for each fragment drawn
glStencilFunc(GL_ALWAYS, 1, 0xFF); // All fragments should pass the stencil test
glStencilMask(0xFF); // Enables writing to the stencil buffer
DrawContainer();

// Draws the object outlines
glStencilFunc(GL_NOTEQUAL, 1, 0xFF); // Fragments that equal the previously draw objects will not pass the stencil test, so the outline
                                     // will only be drawn on the visible borders of the object.
glStencilMask(0x00); // Disables writing to the stencil buffer
glDisable(GL_DEPTH_TEST); // Prevents the outline from beating out outlined objects' fragments.
outlineShader.use();
DrawScaledUpContainer();

// Resets the stencil and depth testing settings back to default
glStencilMask(0xFF);
glStencilFunc(GL_ALWAYS, 1, 0xFF);
glEnable(GL_DEPTH_TEST);
```

Stencil Buffer after DrawContainer() (pixels not to scale)



Stencil Buffer after DrawScaledUpContainer() (pixels not to scale)



Better Object Outlining

Friday, April 22, 2022 5:52 PM

Use the same code and fragment shader as before, but don't scale the outline model in the render loop. Instead, scale it in the vertex shader.

outline.vert

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
   vec3 FragPos = vec3(model * vec4(aPos, 1.0));
   vec3 Normal = normalize(mat3(transpose(inverse(model))) * aNormal);
   gl_Position = projection * view * model * vec4(aPos + aNormal * 0.05, 1.0); // scales vertex position by 1.05 along the normal
}
```

Difference

Previous method

Current method

Issue with this method

NOTE: This solution is still very primitive and still does not create a perfect outline (sharp changes in shape will have noticable gaps in the highlighting, which could be fixed by smoothing the normals of the vertices so each vertex's normal direction vector is the normalized sum of all adjacent vertices' normals).

- Creating an outline for an object is easily surpassed in performance using some kind of post-processing effect because for each object you want to outline, you have to re-draw all the triangles of that object to create the outline model (you can see this by including `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)` in your code). This means that, if your model has 1 million triangles, you have to draw another 1 million triangles to create the outline effect. Very inefficient.