FACE CULLING

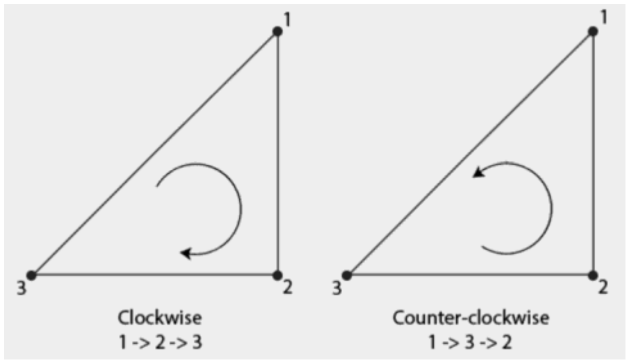
Currently, we are drawing every face of an object, with no regard for whether it is visible or not. This is horribly inefficient. We can cut down on the number of fragment shader calls by simply discarding the triangles we can't see, and then drawing them when they would need to be rendered on the screen. This is known as face culling.

- Example: Think about a cube. A cube has 6 faces, but you can (typically) only ever see a maximum of 3 of them, or a minimum of 1 of them. By not rendering the non-visible faces of the cube, you can cut your fragment shader calls down to 16.67% - 50% of the amount of runs you would have had by drawing every face. This is a HUGE performance boost.

So, how do we know if a face is not visible from the viewer's point of view?

- If a face is **front-facing**, meaning its front face is towards the viewer, then it is rendered.
- If a face is **back-facing**, meaning its back face is towards of the viewer, then it is discarded.

OpenGL analyzes the **winding order** of the vertex data to determine whether a triangle is front-facing or back-facing.



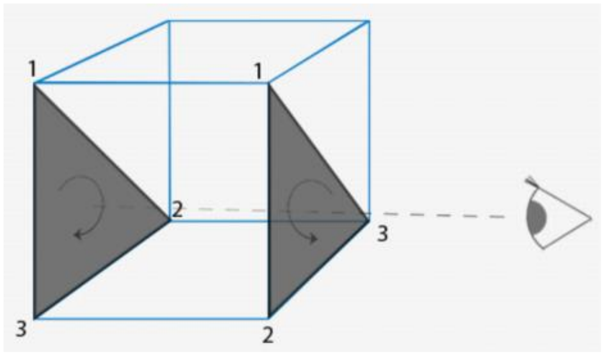
Winding Order

When we define a set of triangles, we're defining them in a certain winding order; that is, either **clockwise** or **counter-clockwise** orientation.

- Each triangle consists of 3 vertices and we specify those 3 vertices in a winding order as seen from the center of the triangle (shown in the image to the right).
  - We first define vertex 1 and, from there, we can choose whether the next vertex is 2 or 3. This choice defines the winding order of the triangle.

OpenGL uses the winding order when rendering primitives to determine if a triangle is front-facing or back-facing.

- By default, triangles with a counter-clockwise winding order are processed as front-facing triangles.
  - **NOTE:** This can vary depending on the graphics API you use; some APIs (like Direct3D) consider triangles with a clockwise winding order as front-facing.
- All 3D applications export models with consistent winding orders (CCW by default).
- When defining your vertex order, you visualize the corresponding triangle if it was facing you, so each triangle that you're specifying should be counter-clockwise as if you're directly facing that triangle.

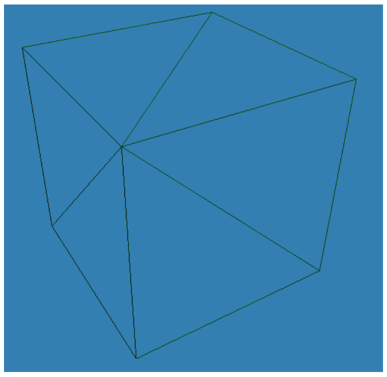


Winding order is calculated automatically at the rasterization stage (after the vertex shader has already run).

If you look at the image on the right, the vertices of the triangles at the other side of the cube are now rendered in such a way that their winding order becomes reversed. The result is that the triangles we're facing are seen as front-facing triangles and the triangles at the back are seen as back-facing triangles.

Face Culling

The counter-clockwise winding ordered vertex data for a cube can be found here: Counter-Clockwise Vertex Data



To enable face culling, use `glEnable(GL_CULL_FACE)`.

- By default, face culling discards all the faces that are not front-facing.
  - You can confirm this by moving the view inside a cube. It will appear as if the cube is invisible, from the inside.
  - This saves over 50% of performance on rendering fragments if OpenGL decides to render the back-facing primitives first. Otherwise, depth testing would've discarded those faces already.
  - **NOTE:** This only really works with closed shapes. Face culling should be disabled before drawing open shapes (e.g. grass, hair, etc. that don't have volume), where the front and back faces should be visible.

You can define what type of face you want to cull using `glCullFace`.

- `glCullFace(GL_BACK)` culls only the back faces (default).
- `glCullFace(GL_FRONT)` culls only the front faces.
- `glCullFace(GL_FRONT_AND_BACK)` culls both the front and back faces.

You can also tell OpenGL how to interpret the what a front-facing primitive is using `glFrontFace`.

- `glFrontFace(GL_CCW)` tells OpenGL that triangles with a counter-clockwise winding order are front-facing triangles (default).
- `glFrontFace(GL_CW)` tells OpenGL that triangles with a clockwise winding order are front-facing triangles.

EXERCISES

1. Can you redefine the vertex data by specifying each triangle with a clockwise winding order and then render the scene with clockwise triangles set as the front faces?

Counter-Clockwise Vertex Data

Friday, April 22, 2022 11:46 PM

```
/*
    Remember: to specify vertices in a counter-clockwise winding order you need to visualize the triangle
    as if you're in front of the triangle. From that point of view is where you set their order.

    To define the order of a triangle on the right side of the cube for example, you'd imagine yourself looking
    straight at the right side of the cube, and then visualize the triangle and make sure their order is specified
    in a counter-clockwise order. This takes some practice, but try visualizing this yourself and see that this
    is correct.
*/

float cubeVertices[] = {
    // Back face
    -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, // bottom-left
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f, // top-right
     0.5f, -0.5f, -0.5f,  1.0f, 0.0f, // bottom-right
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f, // top-right
    -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, // bottom-left
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, // top-left
    // Front face
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, // bottom-left
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f, // bottom-right
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f, // top-right
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f, // top-right
    -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, // top-left
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, // bottom-left
    // Left face
    -0.5f,  0.5f,  0.5f,  1.0f, 0.0f, // top-right
    -0.5f,  0.5f, -0.5f,  1.0f, 1.0f, // top-left
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, // bottom-left
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, // bottom-left
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, // bottom-right
    -0.5f,  0.5f,  0.5f,  1.0f, 0.0f, // top-right
    // Right face
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f, // top-left
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f, // bottom-right
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f, // top-right
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f, // bottom-right
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f, // top-left
     0.5f, -0.5f,  0.5f,  0.0f, 0.0f, // bottom-left
    // Bottom face
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, // top-right
     0.5f, -0.5f, -0.5f,  1.0f, 1.0f, // top-left
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f, // bottom-left
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f, // bottom-left
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, // bottom-right
    -0.5f, -0.5f, -0.5f,  0.0f, 1.0f, // top-right
    // Top face
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, // top-left
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f, // bottom-right
     0.5f,  0.5f, -0.5f,  1.0f, 1.0f, // top-right
     0.5f,  0.5f,  0.5f,  1.0f, 0.0f, // bottom-right
    -0.5f,  0.5f, -0.5f,  0.0f, 1.0f, // top-left
    -0.5f,  0.5f,  0.5f,  0.0f, 0.0f  // bottom-left
};
```