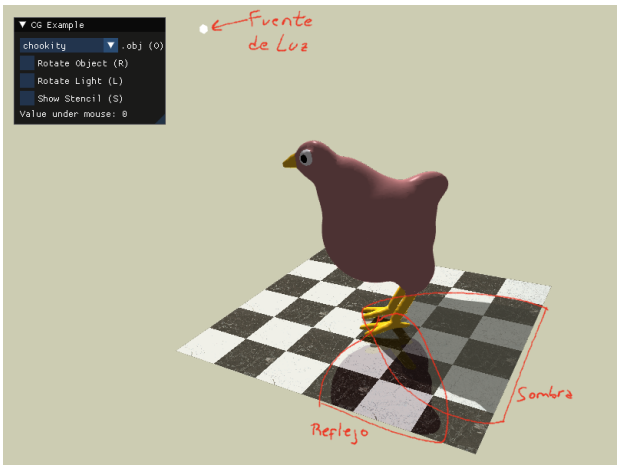
Computación Gráfica - TP: Planar Shadows

1. Resumen de tareas

1. Utilizar el *stencil buffer* para renderizar correctamente el reflejo y la sombra del objeto sobre el piso.

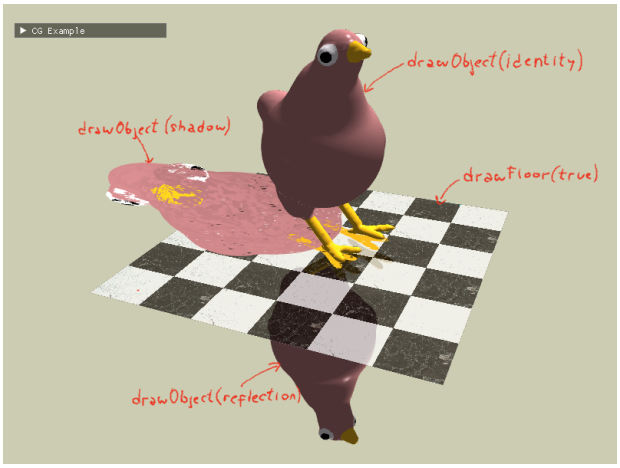
2. Consigna detallada

El objetivo de este práctico es simular dos efectos que el modelo de iluminación local que utilizamos habitualmente no logra capturar: sombras y reflejos.



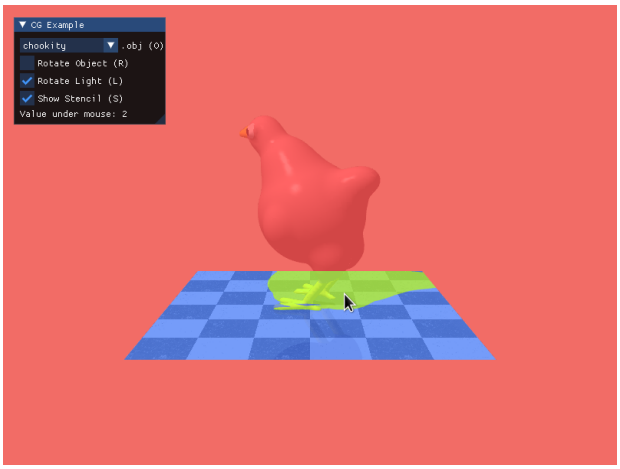
Para ello el código inicial ofrece una función `vod drawObject(const glm::mat4 &m)` que dibuja al objeto, aplicándole una transformación adicional (además de la rotación) que recibe como argumento (la matriz `m`). Utilizando esta función con las matrices adecuadas, se pueden obtener versiones espejadas (respecto al plano del piso) y aplastadas (contra el plano del piso) del objeto. La matrices para ello ya están implementadas y se obtienen a partir de las funciones `getReflectionMatrix()` y `getShadowMatrix()`. Si se guarda la matriz que retorna la primera función en una variable `reflection`, la matriz que retorna la segunda en otra variable `shadow` y la matriz identidad en una tercera variable `identity`, la siguiente captura muestra el resultado de invocar a `drawObject` con cada una de ellas.

Además, una función `drawFloor(bool light_on)` dibujar el piso. El argumento de la función indica si hay que dibujar el piso como si estuviera iluminado (cuando recibe `true`) o como si estuviera en sombra (cuando recibe `false`).



Para resolver el práctico deberá utilizar el *stencil buffer* para que la versión reflejada del objeto solo se dibuje donde coincide con el piso, y el piso se dibuje iluminado donde no coincide con la versión aplastada del objeto (que no debe verse), y en sombre en donde sí.

Como ayuda, el práctico cuenta con dos formas de visualizar el contenido del *stencil buffer*. La opción *show stencil* de la ventana de configuración hace que se tiña el resultado con diferentes colores de acuerdo a los diferentes valores del *stencil buffer*:



Además, el valor mostrado en *Value under mouse* en la misma ventana, es el valor que guarda el *stencil buffer* para la posición actual del ratón. Puede mover el ratón para explorar el contenido del buffer, o saber cuando lo visualiza a qué valor corresponde cada color.

2.1. Funciones relacionadas a los tests y buffers

Algunas funciones de *OpenGL* para manipular el funcionamiento de los tests y el uso de los buffers asociados:

- `glEnable(GLenum cap)`/`glDisable(GLenum cap)`: Estas funciones sirven, como sus nombres lo indican, para habilitar o deshabilitar muchas cosas (qué depende del argumento `cap`). Por ej, si se le pasa como argumento la constante `GL_STENCIL_TEST` se habilita o deshabilita el test de *stencil*; si se le pasa como argumento

`GL_DEPTH_TEST` se habilita o deshabilita el test del algoritmo del *z-buffer*; y si se le pasa como argumento `GL_BLEND` se habilita o deshabilita el blending (mezcla entre el color de un nuevo fragmento y el que había previamente en el *color buffer*).

- `glDepthFunc(GLenum func)`: Permite definir qué comparación se debe hacer en el test de z (entre el z del fragmento y el almacenado en el *depth buffer*). Por ejemplo, el funcionamiento por defecto equivale a utilizar `glDepthFunc(GL_LESS)`, ya que el comparador `GL_LESS` indica que un fragmento pasará el test cuando su valor sea menor (*less than*) que el almacenado en el buffer.
  - Las funciones posibles son: `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, y `GL_ALWAYS`.
- `glDepthMask(GLboolean b)`: habilita (si recibe `GL_TRUE`) o deshabilita (si recibe `GL_FALSE`) la escritura del *depth buffer*. Es decir, que se puede mantener el test activado, pero evitar que el contenido del buffer se actualice cuando un fragmento lo pasa.
- `glBlendFunc(GLenum sfactor, GLenum dfactor)`: Cada vez que se genera un fragmento, si la etapa de blending está activada, se obtiene el color final del pixel interpolando entre el color del fragmento y el color que había previamente en el buffer. Esta función permite indicar qué valores utilizar como pesos para esa interpolación o mezcla. Lo habitual es usar `glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA)`, lo cual indica que los pesos serán *alpha* y *1-alpha*, tamando como valor de *alpha* al que trae el fragmento(*src*) en su color.
  - Los valores posibles son: `GL_ZERO`, `GL_ONE`, `GL_SRC_COLOR`, `GL_ONE_MINUS_SRC_COLOR`, `GL_DST_COLOR`, `GL_ONE_MINUS_DST_COLOR`, `GL_SRC_ALPHA`, `GL_ONE_MINUS_SRC_ALPHA`, `GL_DST_ALPHA`, `GL_ONE_MINUS_DST_ALPHA`, `GL_CONSTANT_COLOR`, `GL_ONE_MINUS_CONSTANT_COLOR`, `GL_CONSTANT_ALPHA`, y `GL_ONE_MINUS_CONSTANT_ALPHA`.
- `glColorMask(GLboolean r, GLboolean g, GLboolean b, GLboolean a)`: habilita (si recibe `GL_TRUE`) o deshabilita (si recibe `GL_FALSE`) la actualización de cada canal del color buffer. Si un canal está deshabilitado, aunque un fragmento supere todos los tests modificará su valor en el color buffer.
- `glStencilFunc(GLenum func, int ref, unsigned int mask)`: permite configurar la operación a realizar para determinar si un fragmento supera el test de *stencil*. Se comparan el valor de referencia (`ref`) con el almacenado en el buffer, pero previamente enmascarando (producto bit a bit, `&`) ambos valores con la máscara `mask`. La función de comparación se define con `func`. Por ejemplo: `glStencilFunc(GL_EQUAL,1,~0)` indica para pasar el test el valor del buffer debe ser 1 (y en este caso la máscara no influya, ya que `~0` es poner todos los bits en 1).
  - Las funciones posibles son las mismas que para `glDepthFunc`.
- `glStencilOp(GLenum fail, GLenum zfail, GLenum pass)`: Indica qué actualización se debe hacer al valor del *stencil buffer* de acuerdo al resultado de los tests. El primer argumento define la operación a realizar cuando un fragmento no pasa el test de *stencil*, el segundo cuando pasa el test de *stencil* pero no el de *z*, y el tercero cuando pasa ambos.
  - Las operaciones posibles son: `GL_KEEP`, `GL_REPLACE`, `GL_ZERO`, `GL_INCR`, `GL_INCR_WRAP`, `GL_DECR`, `GL_DECR_WRAP` y `GL_INVERT`.
- `glStencilMask(GLuint mask)`: define una máscara que indica qué bits serán actualizados en una operación que modifique al *stencil buffer*. Notar que a diferencia de las otras funciones de máscara (`glColorMask` y `glDepthMask`) donde se enmascaraba todo el canal completo (el argumento era simplemente verdadero o falso), en esta función se define el comportamiento individual de cada bit (el argumento es un entero, no un booleano).

En la mayoría de los casos donde se listaron posibles valores para los argumentos, los nombres de las constantes permiten deducir su significado, pero si necesita mayores detalles puede buscar las funciones en la referencia de *OpenGL*: https://registry.khronos.org/OpenGL-Refpages/gl4/html/indexflat.php.