Computación Gráfica - TP: Outlining

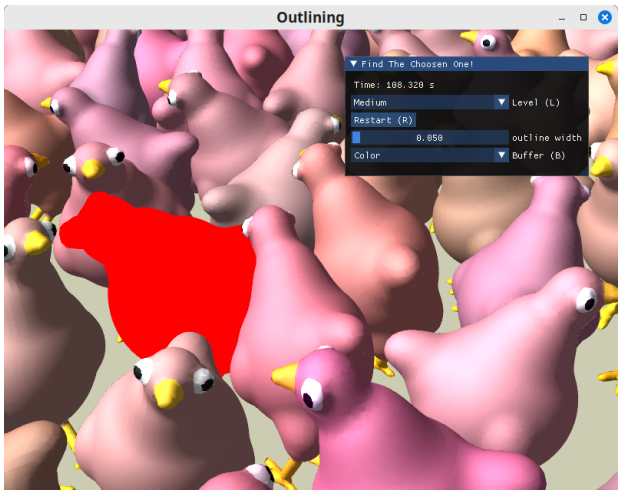
1. Resumen de tareas

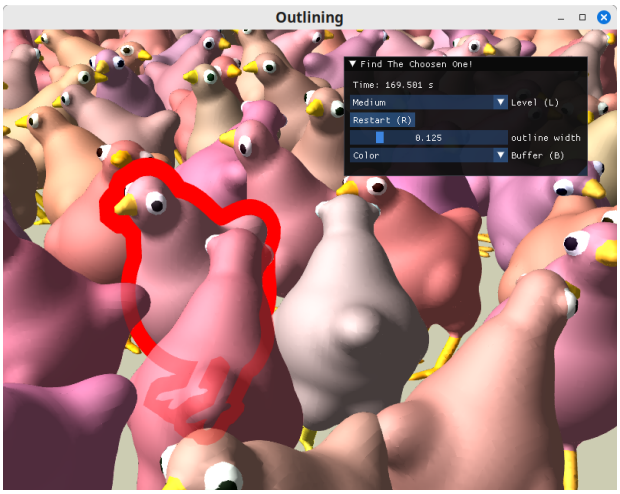
1. Utilizar el *stencil buffer* para generar un *outlining* (resaltado con un color fuerte de la silueta alrededor de un objeto) del objeto seleccionado.
2. Analizar el mecanismo de selección (ya implementado).
3. Analizar el mecanismo de ”inflado” del modelo (ya implementado).

2. Consigna detallada

El código provisto en este TP implementa un juego de dinámica similar al famoso ”¿Dónde está Wally?” (o ”Where is Waldo?”), pero utilizando a la mascota de la cátedra *Chookity* como protagonista. Cuando el juego inicia se posicionan aleatoriamente en escena cientos de copias de *Chookity* casi idénticas, y una especial. El *Chookity* especial (desde ahora ”El Elegido”) tiene los ojos cubiertos con una venda de color lila. El objetivo es encontrar al Elegido en el menor tiempo posible. Cuando el usuario hace doble click sobre un Chookity, si es el incorrecto este se resalta en rojo y se suma un tiempo de penalización. Si es El Elegido, se resalta en verde, el tiempo se detiene y el juego finaliza.

Inicialmente, el código ”resalta” al Chookity seleccionado ”inflándolo” y pintándolo completamente de verde o rojo sólido. **El alumno deberá modificar el código e idear una técnica multipaso que manipulando el Stencil Buffer y/o el algoritmo del Z-Buffer, permita resaltar solamente la silueta del Chookity seleccionado, utilizando un color sólido cuando sea directamente visible, y semitransparente donde esté tapado por otro objeto. **





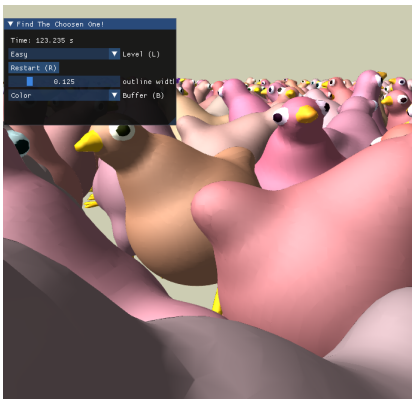
Dentro del código, la función a modificar para corregir el resaltado es `outlineInstance`.

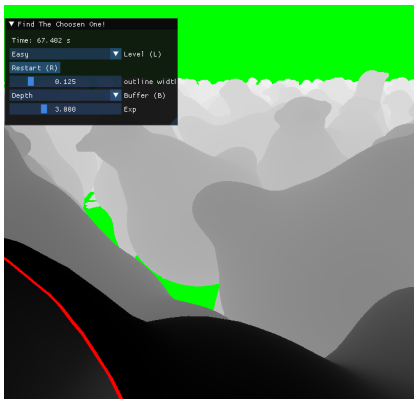
Luego, **el alumno debe analizar el mecanismo de selección implementado (¿cómo se determina sobre cual instancia estaba el mouse al hacer doble click)**. Este algoritmo se encuentra en la función `findSelection`.

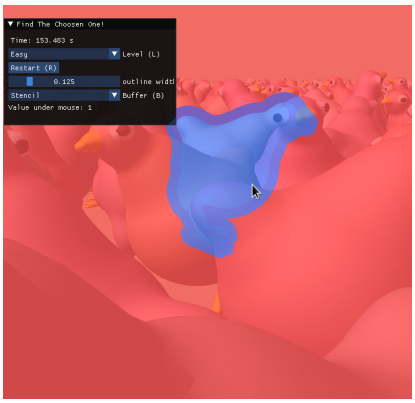
Por último, **deberá además analizar cómo logra el código ”inflar” el modelo** para que el resaltado sobresalga en todas las direcciones respecto a la silueta del original (lo que se controla con la variable `outline_width` del shader `sillhouette`.

En todo momento puede utilizar el combo que dice ”Buffer” para ”visualizar” el contenido de los distintos buffers.

- Cuando seleccione el *stencil buffer*, se superpondrá el contenido del mismo sobre el buffer de color, tiñendo cada pixel con un color que dependerá de su valor en el *stencil buffer*. Así podrá visualizar las distintas zonas o máscaras que genere en el mismo. Además, podrá mover el cursor del ratón por la imagen y el programa indicará con un texto debajo del combo el valor que corresponda al pixel que señale el ratón.
- Cuando seleccione el *z-buffer*, verá un imagen en escala de grises, ya que en este caso el contenido del Z-Buffer se interpreta como una imagen en escala de grises. Los valores más oscuros corresponden a fragmentos cercanos a la cámara, y los más claros a los lejanos. Dispondrá de un slider adicional ”exp” para utilizar el valor de Z elevado a un exponente para generar el nivel de gris (similar a una corrección gamma) y acentuar así la diferencia entre valores cercanos. Además, los pixeles con valores muy pequeños (*Z-Near* o muy cernanos) se resaltarán en rojo, y los más lejanos (*Z-Far*) se en verde.







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.

Finalmente, es importante mencionar que en *GLSL*, existe la palabra clave `discard` para indicar que un fragmento debe descartarase. Con este comando puede, por ejemplo, implementarse un equivalente al obsoleto (*deprecated*) *alpha-test* (ej: `if (fragColor.a<1.f) discard;`);