Computación Gráfica - TP: Aplicación de Texturas

1. Resumen de tareas

1. Implementar la generación automática de coordenadas de textura utilizando en cada caso el método que considere más adecuado para producir los efectos deseados (lens flares y el fondo de la escena).
2. Analizar los problemas encontrados en el resultado y proponer (no es necesario implementar) posibles soluciones.
3. Implementar un shader que permita *visualizar* las coordenadas de textura.

2. Consigna detallada

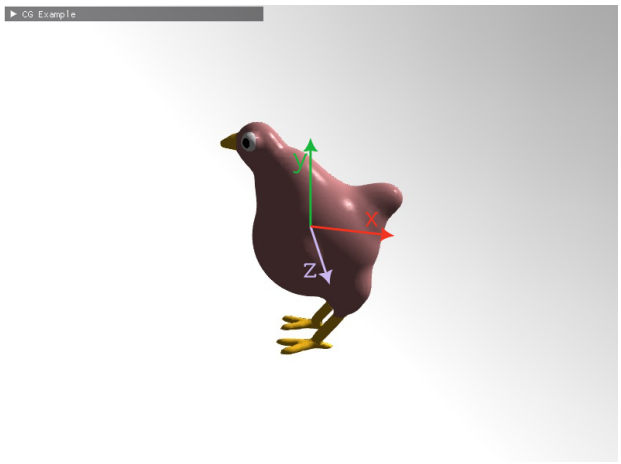
2.1. Generación de las coordenadas de textura para el ambiente

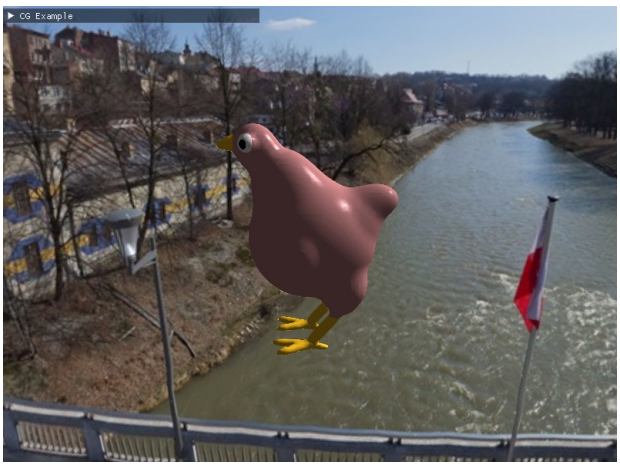
El código inicial carga en memoria y renderiza dos *modelos* (dos mallas en dos instancias de `Model`). La primera, la que nos interesa para este practico, corresponde a una esfera (*sphere.obj*), la cual se va a renderizar de modo que envuelva el resto de la escena (la cámara se ubica dentro para ver los demás objetos), y se le deberá aplicar una textura para crear la sensacion de estar dentro de un ambiente mas complejo (se usa como imagen de fondo).

Esta tecnica es conocida comúnmente como *sky box*, ya que usualmente se realiza con texturas de cubos (*sky dome* para el caso de nuestra esfera).

El resto de los *modelos* se renderizan dentro de este *ambiente* y puede intercambiarse si se desea (chookity, suzzane, etc).

En los datos de entrada (el archivo *sphere.obj*) se encuentran las coordenadas de los vértices, las normales de los mismos y las conectividades (los triángulos); pero **la información de entrada no contiene coordenadas de textura para los vértices. El alumno debe generarlas (es decir, calcularlas con algún método automático en función de las coordenadas de los vértices)**, para poder aplicar la textura que se carga desde el archivo *sky_dome.jpg*. El resultado esperado es el siguiente:





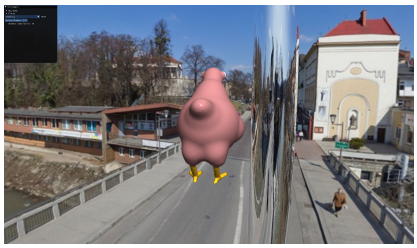
La función a completar para este paso se llama `generateTextureCoordinatesForSkyDome` y esta definida en *DrawScene.cpp*. La función recibe como argumento de entrada un vector con los nodos (coordenadas) de la malla

(`std::vector<glm::vec3>`), y deben retornar un nuevo vector del mismo tamaño con las coordenadas de textura para dichos nodos (`std::vector<glm::vec2>`). Cada elemento del vector de salida se corresponde con el elemento del vector de entrada de la misma posición (índice).

Si realiza una primera implementación que no resuelve correctamente el problema, puede pasar al paso 3 (Visualización de las Coordenadas de Textura) para lograr así un mejor mecanismo de depuración que le permita entender los problemas.

2.2. Problema esperado

Es probable que luego de implementar la función indicada en el punto anterior, se encuentre con el siguiente problema:

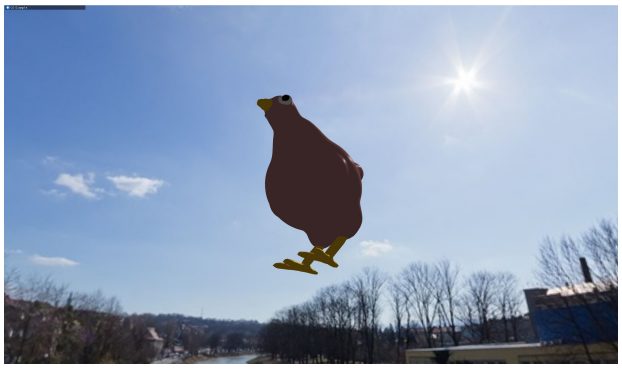


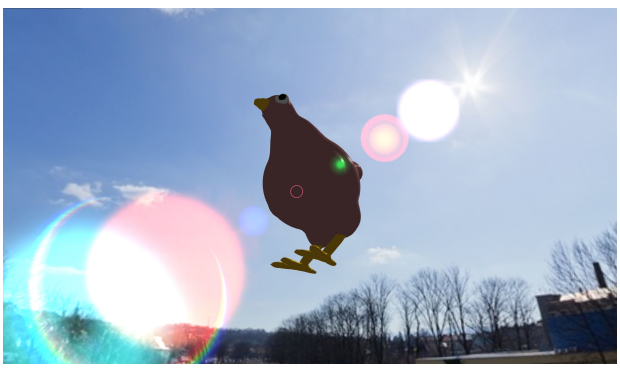
En una parte se encuentra repetida y aplastada toda la textura de la esfera. **Analice el problema para entender la causa del mismo y proponga una posible solución (no debe implementarla**, la solución requeriría trabajo adicional fuera de la función modificada en el primer paso).

Ayuda: Analice el problema una vez implementado el último punto para visualizar que ocurre con las coordenadas de textura.

2.3. Generación de las coordenadas de textura para el efecto Lens Flare (Destello de lente)

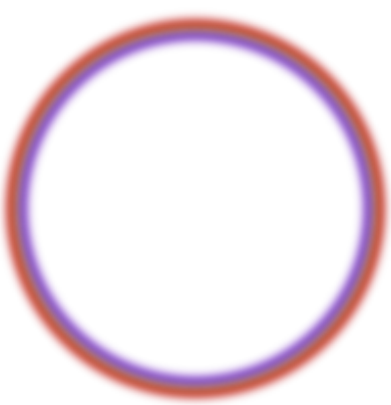
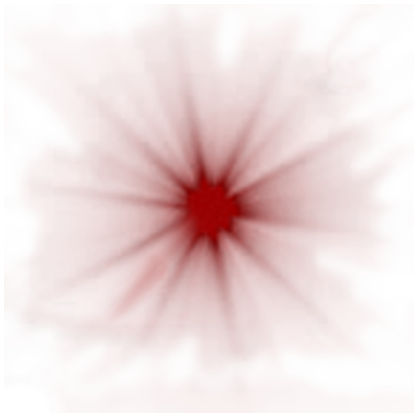
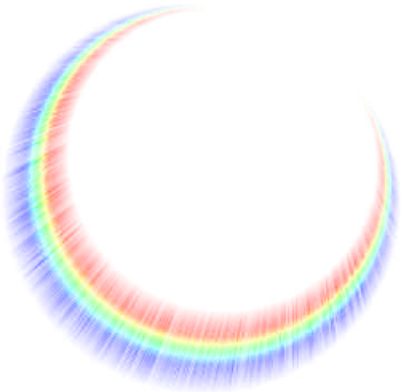
En este practico se buscara reproducir, de manera sencilla, un efecto visual conocido como *lens flare o destello de lente*. Es un efecto de finalidad puramente estético/artístico que se genera cuando la luz directa entra por el objetivo de una cámara, generando reflejos internamente en la lente. El resultado sera como en la siguiente imagen:



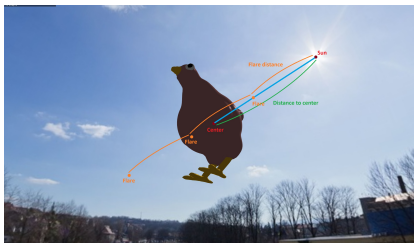


Estos reflejos se renderizan de forma individual (clase *Flare*), cada uno sobre un *billboard* a pantalla completa. Es decir, un quad que cubre toda la pantalla[1] y está alineado siempre de frente a la cámara. Sus vértices, en el espacio NDC son: {-1.f, -1.f, 0.f}, { 1.f, -1.f, 0.f}, { 1.f, 1.f, 0.f} y {-1.f, 1.f, 0.f}, en ese orden.

Las siguientes imágenes corresponden a algunas de las texturas de los flares:

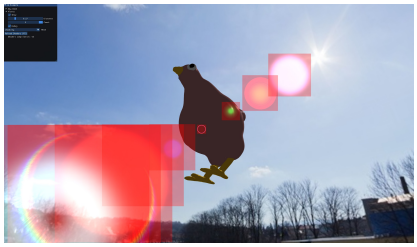
Desde la posición de la luz en la pantalla (*screen_coordinates*), y en dirección al centro de la misma, se determina la posición (y tamaño) de cada *flare* en secuencia, hasta una cantidad fija.



La función a completar para este paso se llama `generateTextureCoordinatesForFlare` y está definida en *Flare.cpp*. La función recibe como argumento de entrada el tamaño en pixeles (*int*) y la posicion (x,y) en la pantalla (*glm::vec2*) del centro del *flare* (los puntos naranja en la figura), y debe retornar un vector con las coordenadas de textura para los nodos del *billboard* (`std::vector<glm::vec2>`). El *billboard* es un simple quad en pantalla, solo se necesita la coordenada de textura de cada uno de los 4 vertices (esquinas).

Ayuda: En la GUI, dentro del menú `Flares`, ademas de las propiedades para manejar el display (distancia y cantidad de *flares*), se cuenta con una opción extra que carga un shader auxiliar, el cual muestra la posición donde deberían verse los *flares* en la pantalla (para ajustar a ojo).

[1] Se usa un quad que cubre toda la pantalla y se ”ubica” el flare sobre la misma moviendo la textura solamente porque este es un TP de la unidad de texturas, solo con el fin de evaluar el mapeo. En una aplicación real sería más razonable dibujar un quad cuya posición y tamaño coincida con el flare (y entonces el mapeo de la textura sería trivial).



2.4. Visualización de las Coordenadas de Textura

El práctico cuenta con una opción (la tecla *C* o el check ”Show tex coords” en la GUI) para utilizar un shader alternativo (implementado en los archivos *tex_coords.vert* y *tex_coords.frag*). Tomando esto como base, **modifique el fragment shader para lograr ”visualizar” de alguna forma las coordenadas de textura asignadas a cada fragmento**.

Ayuda: recuerde que el valor a utilizar siempre estará entre 0 y 1, pero los valores *s* y *t* que recibe un fragmento podrían no estarlo (se ajustan mediante los mecanismos de *clamp* o *repeat* al consultar la textura).

Recuerde que puede modificar libremente los shaders y recargarlos en la interfaz (atajo F5) sin tener que reiniciar el programa.