Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Estado de México
Escuela de Ingeniería y Ciencias
Departamento de Computación

Proyecto 04. Buffers y Shaders. Gráficas computacionales.

Apegándome al Código de Ética de los Estudiantes del Tecnológico de Monterrey, me comprometo a que mi actuación en este examen esté regida por la honestidad académica. Acepto: Isain Cuadra

Profesor: Oriam Renan De Gyves López	Clave y grupo: TC3022. 1
Alumno: Isain Cuadra Rivas	Matrícula: A01375997

Lee cuidadosamente todas las instrucciones antes de comenzar a resolver cada uno de los problemas.

Fecha de entrega: 24 de marzo, antes de las 23:59 horas.

NOTA MUY IMPORTANTE: Este proyecto debe ser elaborado de manera individual. Puedes trabajar de manera colaborativa para resolver dudas conceptuales, pero el código debe ser realizado únicamente por ti. Recuerda citar correctamente todas las fuentes de inspiración que ocupaste para la realización del mismo. Cualquier indicio de copia, trampa o fraude será penalizado con 1 de calificación y será reportado al comité de integridad académica. Se penalizará tanto al copiado como al copiador.

Documentación:
Manual de referencia de C++: http://www.cplusplus.com/reference/
Documentación de OpenGL: http://docs.gl/

Deberán leer:

`glCreateShader`: http://docs.gl/gl4/glCreateShader
`glDeleteShader`: http://docs.gl/gl4/glDeleteShader
`glShaderSource`: http://docs.gl/gl4/glShaderSource
`glCompileShader`: http://docs.gl/gl4/glCompileShader
`glGetShaderiv`: http://docs.gl/gl4/glGetShader
`glGetShaderInfoLog`: http://docs.gl/gl4/glGetShaderInfoLog

`glCreateProgram`: http://docs.gl/gl4/glGenVertexArrays
`glDeleteProgram`: http://docs.gl/gl4/glGenVertexArrays
`glAttachShader`: http://docs.gl/gl4/glGenVertexArrays
`glBindAttribLocation`: http://docs.gl/gl4/glGenVertexArrays
`glLinkProgram`: http://docs.gl/gl4/glGenVertexArrays
`glGetProgramiv`: http://docs.gl/gl4/glGetProgram
`glGetProgramInfoLog`: http://docs.gl/gl4/glGetProgramInfoLog

`glUseProgram`: http://docs.gl/gl4/glGenVertexArrays
`glGetUniformLocation`: http://docs.gl/gl4/glGenVertexArrays
`glUniform`: http://docs.gl/gl4/glGenVertexArrays
`glDrawArrays`: http://docs.gl/gl4/glGenVertexArrays
`glDrawElements`: http://docs.gl/gl4/glGenVertexArrays

Capítulo 6 del libro OpenGL Superbible (7ma edición): Shaders and programs.
https://www.safaribooksonline.com/playlists/13c37da7-e4f3-4940-8680-b8055f3824b8

Evaluación
El proyecto será evaluado utilizando los siguientes criterios:

100	El proyecto cumple con todos los requerimientos.
1	El programa fuente contiene errores sintácticos.
1 - 99	El programa produce errores al momento de correrlo.
Falta a la integridad académica	La solución es un plagio.

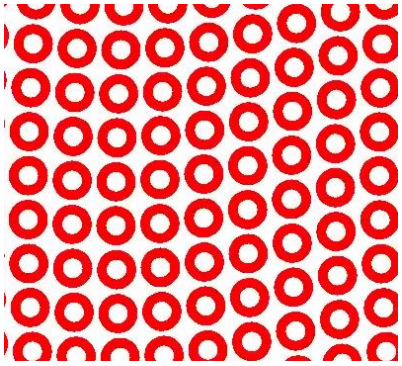
Nota: *Todas las escenas deberán estar cargadas en el* ***`scene_manager`****.*
Nota: *encontrarás videos / imágenes representativas de los resultados esperados en el grupo.*

I. **(25 puntos)** Crea un *vertex shader*, llamado **circle_grid.vert**, que mezcle los programas **grid.vert** y **circle.vert** hechos en clase, de tal manera que se muestre en pantalla una malla de círculos animada. Crea la escena **scene_circle_grid** para mostrar tu resultado.

a. No debes utilizar atributos (*vertex streams*).
b. Las posiciones de los vértices deben ser calculados en el *vertex shader*.
c. La posición de cada elemento en la malla deberá estar animada como en el programa **grid.vert**.
d. Deberás utilizar la primitiva **GL_TRIANGLE_STRIP** para visualizar la malla de círculos.

Fuente de inspiración para los primeros 2 ejercicos

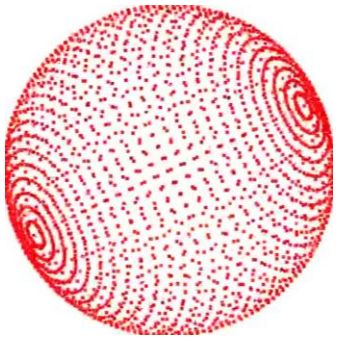
Bermudez Aldo



Representación visual del resultado final del ejercicio 1

II. **(25 puntos)** Crea un *vertex shader*, llamado **sphere.vert**, que utilice coordenadas polares para dibujar una esfera en 3 dimensiones. Crea la escena **scene_sphere** para mostrar tu resultado.

a. No debes utilizar atributos (*vertex streams*).
b. Las posiciones de los vértices deben estar calculadas en el *vertex shader*.
c. Deberás animar la rotación de la esfera utilizando matrices de rotación. La rotación debe ocurrir en al menos 2 ejes (XZ, XY, YZ).
d. Utiliza la primitiva **GL_POINTS** para visualizar la esfera.

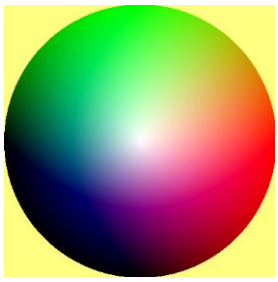


Representación visual del resultado final del ejercicio 2

III. **(25 puntos)** Utilizando la primitiva más conveniente, **genera los atributos** (*vertex streams*) necesarios para dibujar un círculo de radio 1. Crea la escena **scene_circle** para mostrar tu resultado. El círculo deberá tener las siguientes características:

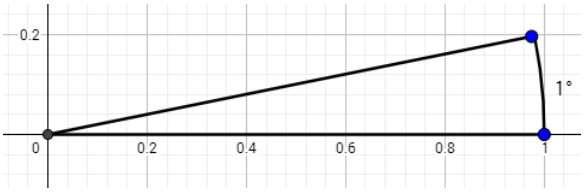
Para este y el ejercicio siguiente tuve una fuente de inspiracion, mi fuente fue:

Alquizer Antonio

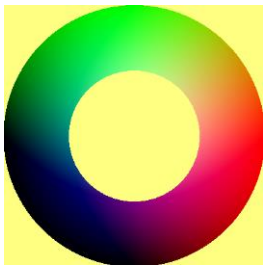


Cada uno de los vértices sobre la circunferencia del círculo deberá tener un color diferente. Los colores deberán difuminarse. No es necesario que los colores sean los mismos que en esta imagen, pero si debe existir variación entre cada uno de los vértices.

El círculo está compuesto por una serie de triángulos. Todos los triángulos tienen un vértice en el centro del círculo. El ángulo que se forma por los dos lados, de cada uno de los triángulos, que no están sobre el perímetro del círculo debe ser de exactamente 1°, como se muestra a continuación:



Deberás utilizar un *vertex shader* básico que únicamente asigne el valor del atributo de posición a la variable *gl_Position*. Adicionalmente, crea y utiliza un *fragment shader* que no dibuje los pixeles de la geometría desde el centro de la ventana (0.5, 0.5) hasta un radio de 0.25, como se muestra en la siguiente imagen:



Representación visual del resultado final del ejercicio 3

Utiliza la instrucción `discard` para detener el proceso de dibujado del pixel. La siguiente información concerniente a esta instrucción proviene de la documentación de GLSL:

discard. This keyword can only be used in Fragment Shaders. It causes the fragment shader's results to be discarded and ignored by the rest of the pipeline. That fragment's value(s) will not be written to the framebuffer.
Using discard is different from return from main. Executing return still means that the values written to the shader outputs will be used by the rest of the pipeline. Executing discard means that they will not.

Esto significa que al usar `discard` en un pixel en específico, este ya no es procesado por el pipeline de OpenGL y no se dibuja en pantalla. Puedes utilizar esta instrucción de la siguiente manera:

```
if (/*distancia al centro de la ventana es menor que 0.25*/)
      discard;
```

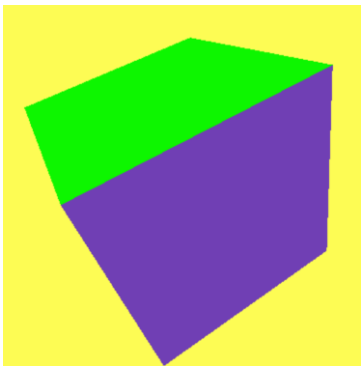
IV. **(25 puntos)** Para este ejercicio, crea la escena **`scene_cube`** para mostrar tu resultado. La escena tendrá los siguientes elementos:

Crear y dibujar un cubo que cumpla con las siguientes características:

a. El atributo de posición debe tener 3 componentes `(x, y, z)`. Es una escena en 3 dimensiones.
b. La longitud de todos los lados del cubo debe ser de 6 unidades.
c. En coordenadas de objeto, el punto `(0, 0, 0)` representa el centro del cubo.
d. Cada cara del cubo debe tener un color diferente. Utilizar el mínimo de vértices necesarios para lograrlo.
e. Todos los atributos se generan en aplicación (CPU) y son enviados a la tarjeta de video usando *buffers*.
f. Utiliza un buffer de índices para optimizar recursos.
g. Utiliza **`GL_TRIANGLES`** como primitiva para dibujar el cubo.

Configura la escena de la siguiente manera:

a. El cubo debe estar posicionado en la coordenada `(0, 0, 0)`. (*Matriz de modelo*)
b. La cámara debe estar posicionada en la coordenada `(0, 0, 10)`. (*Matriz de vista*)
c. La cámara debe usar una proyección en perspectiva con los siguientes valores: (*Matriz de proyección*)
    i. Distancia al plano de corte cercano: 1 unidad
    ii. Distancia al plano de corte lejano: 1000 unidades
    iii. Ángulo de apertura de la cámara (*field of view*): 60° (grados)
    iv. Aspect ratio acorde a las dimensiones de su ventana (cambiar las dimensiones de la ventana no debe cambiar las proporciones del cubo).
d. El cubo debe estar rotando alrededor de sus 3 ejes coordenados. (*Matriz de modelo*)
    Cada segundo debe rotar:
    i. 30° en el eje X
    ii. 60° en el eje Y
    iii. 30° en el eje Z



Representación visual del resultado final del ejercicio 4