UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

PROYECTO FORMATIVO

"Creación de Videjuego 3D"

ELABORADO POR:
Luzquiños Agama, Steve Anthony
Vicente Alva, Gabriel

ASESOR:
Espejo Delzo, Juan Carlos

LIMA-PERÚ

2020

Índice general

Capítulo 1

Introducción

1.1. Motivación

Los videojuegos nos acompañan desde más de 50 años, siempre despertando emociones en propios y extraños, con una industria que actualmente mueve cientos de millones de dólares por año, todos los miembros de nuestra generación conocemos alguno y tenemos alguno preferido; así mismo, aquellos que nos hemos adentrado en el mundo de la computación y nos gusta el desarrollo alguna vez hemos fantaseado con ser desarrollar algún videojuego AAA [1]. Llegados a este punto de la carrera, donde ya tenemos las herramientas "suficientes" como para animarnos a intentar cumplir una de nuestras fantasías de jóvenes, surge la siguiente pregunta, ¿por qué no? Es por esto que nos hemos aventurado en intentar replicar dos juegos clásicos: *Star Fox (1993)*[2] y *Asteroids* [3] dentro de la dinámica de un *endless runner* [4].

1.2. Objetivos

- Realizar un videojuego 3D con la temática de *Star Fox* y la jugabilidad de *Asteroids* (tridimencionalizada).
- Adaptar esta combinación de videojuegos a la dinámica de un *endless runner*.
- Otorgar movimiento al personaje principal por medio de transformaciones lineales y manejo de choques.
- Crear un sistema de puntaje que se base en la destrucción de enemigos (asteroides).

1.3. Estructura de la memoria

En primer lugar, en el capítulo 2, se establece los elementos de las transformaciones lineales y rotaciones; así como los de las colisiones.

En segundo lugar, en el capítulo 3, se ven las dificultades técnicas implicadas en el desarrollo.

Por último, en el capítulo 4, se revisan las soluciones empleados a los problemas planteados anteriormente. Así mismo, capturas de pantalla del resultado final logrado luego de concluido el desarrollo.

Capítulo 2

Fundamento teórico

El presente capitulo es preliminar: tiene por objetivo introducir conceptos y propiedades necesarias para el desarrollo del siguiente trabajo. Para una detallada introducción a las transformaciones lineales se puede consultar el capítulo 4, sección 4.2 de [5]. Así mismo, también puede consultar el capítulo 9, sección 9.6 de [6]. Adicionalmente, puede encontrar información adicional sobre los cuaterniones aquí [**?**]. Finalmente, para una detallada introducción a las colisiones tridimensionales, puede consultar [8], o también [9].

2.1. Transformaciones

Las transformaciones las realizamos usando el plano proyectivo. El plano proyectivo añade puntos .[en] el infinito.[a] un plano, haciendo que dos líneas paralelas siempre tengan una intersección. Podemos hacer esto con $\mathbb{R}^2$ y $\mathbb{R}^3$ añadiendo una coordenada que valdrá 1 para cada punto. En los puntos infinitos valdrá 0.

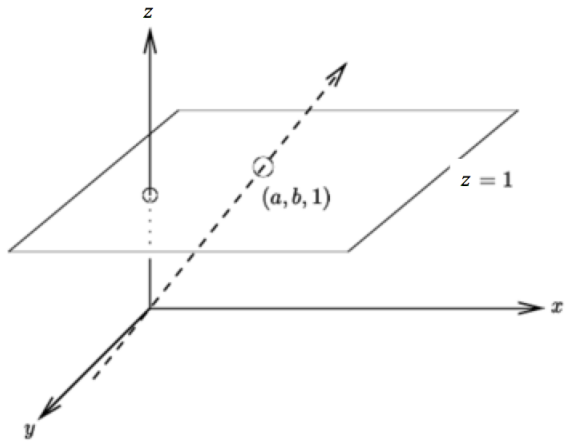

Figura 2.1: Plano proyectivo

Con esto las funciones de traslación, escala, rotación y otras serán transformaciones lineales. Por ejemplo, la transformación de traslación estará dada por:

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Esta matriz se multiplica con el vector a trasladar.

2.2. Cuaterniones

Son una extensión similar a los números complejos en la que tenemos tres unidades imaginarias i, j, k tales que $i^2 = j^2 = k^2 = -1$. Un cuaternión tiene la forma $a + bi + cj + dk$. La parte imaginaria puede verse como un vector, representando todo el elemento como $a + v$. La multiplicación de cuaterniones se obtiene usando la propiedad distributiva. Las rectas que pasan por un cuaternión q y el origen caracterizan rotaciones en 3 dimensiones.
Para rotar el vector v alrededor del vector u:

$$q = e^{\frac{\theta}{2}u} = cos\frac{\theta}{2} + sin\frac{\theta}{2}u$$

$$Rv = qvq^{-1}$$

Así, los objetos del juego tendrán un vector para la posición y un cuaternión para la orientación.

2.3. Detección de colisiones

La detección de colisiones es un método utilizado por videojuegos para detectar si dos objetos han colisionado, esta puede llevarse a cabo píxel a píxel, por área, o por superficie. En nuestro caso, hemos implementado la colisión de superficies en base a la distancia, asumiendo que las superficies que colisionarán serán esferas.

Se obtiene la diferencia entre los centros de ambos objetos y hallamos el módulo del resultado. Para objetos de tamaño considerable esta distancia debe ser menor a la suma de sus radios. Además, en el caso de un proyectil aproximándose a un objeto regular, el radio del proyectil puede obviarse, comparando la distancia solo el radio del otro objeto.

Capítulo 3

Planteamiento del problema

El objetivo final del presente trabajo es la realización de un videojuego 3D, basado en una fusión de dos conocidos videojuegos: *Star Fox* y *Asteroids*, para esto hemos optado por realizar un *endless runner* donde nuestro personaje principal será una nave espacial que "enfrentará" una serie de asteorides, los cuales tendrá que esquivar o destruir, con el fin de ganar el juego. El destruir los asteriodes harán que el puntaje del jugador se incremente.

Tal y como se muestra en las imágenes 4.1, nuestro personaje principal estará "atrapado" en mundo sin fin.

Veamos los aspectos técnicos del problema:

- Los objetos tridimensionales pueden ser definidos a través de vectores y se les puede aplicar transformaciones. Sin embargo, graficar dichos objetos requerirá tomar otros factores en cuenta. Por ejemplo, estos necesitarán un color (serie de colores) o una textura (serie de texturas) para su renderizado, además de que estos usarán transformaciones lineales para poder desplazarse en la escena, finalmente, los objetos deben ser capaces de ser destruidos (desaparecer o reubicarse) por el personaje principal.

- Los polígonos deben ser proyectados en el plano, con esto se sabe qué pixeles de la matriz son "coloreados", estos colores también son calculados según la iluminación, y al graficar uno o varios objetos se debe elegir un método para graficar los polígonos sin que se solapen incorrectamente. Esto con el fin de que el jugador debe ser capaz de reconocer la distancia entre los objetos y el personaje principal.

- Para hacer un videojuego en 3D, en cada fotograma se calculan diversas cantidades, entre ellas por supuesto, la posición de los objetos. Todos estos variarán en el tiempo y por interacciones del jugador. Estos calculos deben ser optimizados para que la tasa de fotogramas no destruya la experiencia de usuario, ya que la experiencia nos indica que la CPU puede procesar una cantidad limitada de enemigos en la escena sin pérdida de paquetes o frames (justamente lo que se busca evitar).

Capítulo 4

Solución del problema

4.1. Personaje Principal

El personaje principal de nuestro juego se muestra a continuación.

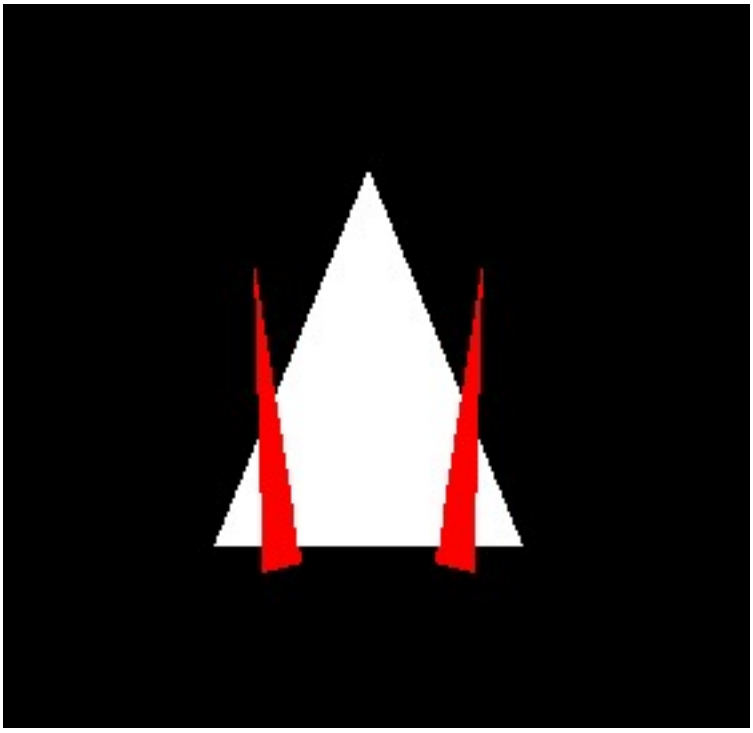

Figura 4.1: Personaje principal

4.2. Diseño de la escena

La nueva escena será desarrollada usando las siguientes librerías de Python: *PyOpenGl* [10].

4.3. Transformaciones

Las transformaciones, y en general las operaciones matemáticas, serán llevadas a cabo usando las siguientes librerías de Python: numpy [11] y pyrr [12].

Capítulo 5

Resultados

- Se logró realizar una fusión de los videojuegos mencionados anteriormente de forma satisfactoria con la dinámica parcial de un *endless runner*.

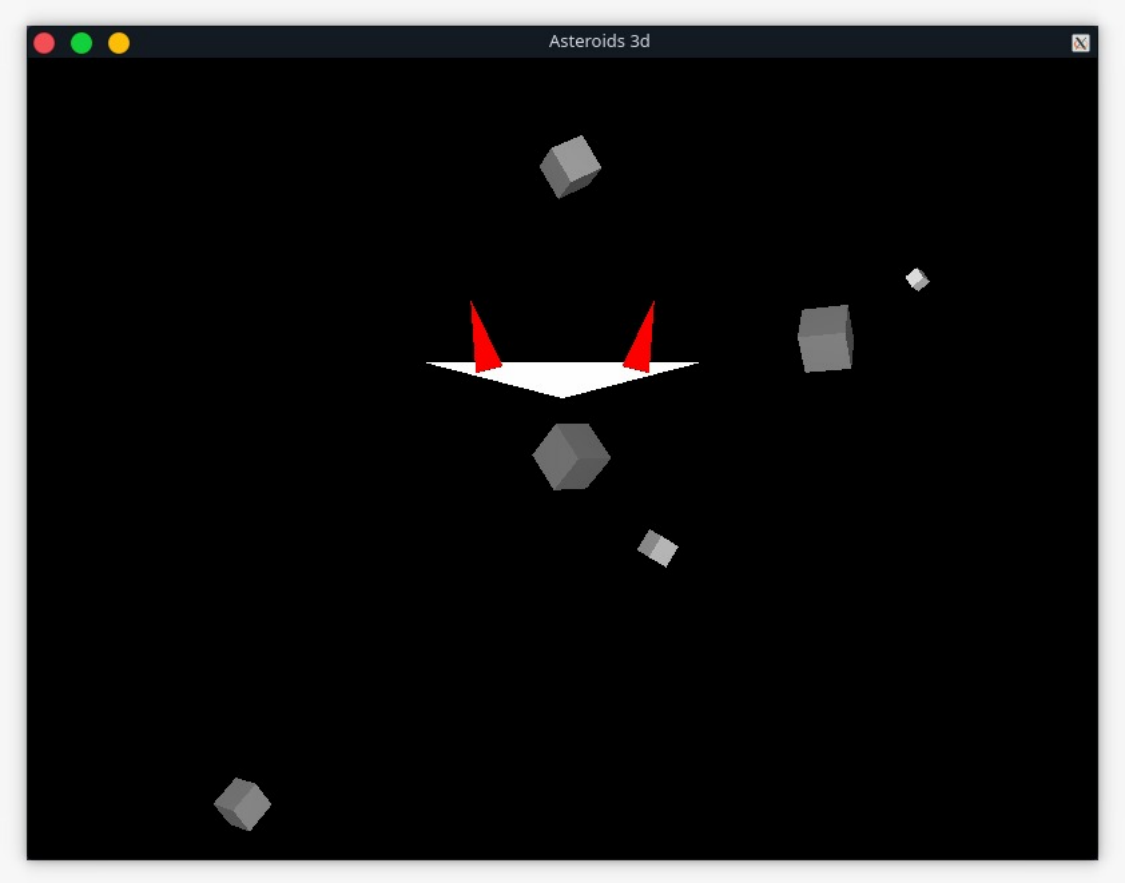

Figura 5.1: Personaje principal

- Se logró "mover" la nave dentro de la escena (generar ilusión de movimiento). Se genera una rotación de la cámara mediante las teclas A, W, S y D, la barra

espaciadora nos permite disparar un proyectil contra los asteroides que se trasladan infinitamente hacia la posición de la cámara.

- Se logró destruir a los asteroides de la escena utilizando colisiones tridimensionales [13].

Bibliografía

[1] Videojuegos triple A.

[2] Star Fox.

[3] Asteroids.

[4] Endless runner.

[5] Steven J. Janke - Mathematical Structures for Computer Graphics (2014, Wiley).

[6] Jhon Vince - Mathematics for Computer Graphics (2017, Springer).

[7] ¿Qué son los cuaterniones y cómo los visualizas? Una historia de cuatro dimensiones.

[8] 3D Collisions detection.

[9] Improved Collision detection and Response.

[10] PyOpenGl.

[11] numpy.

[12] pyrr.

[13] Demo