Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Laboratorio 16

Vídeo Juegos, Programación

Testing y Optimizaciones

1. Pre-Laboratorio

- Investigar sobre los siguientes conceptos:
  1. Quality Assurance.
  2. Software Bug.
  3. Pruebas de Funcionalidad (Functionality Testing)
  4. Pruebas de Compatibilidad (Compatibility Testing)
  5. Pruebas Beta (Beta Testing)
- Investigue sobre el proceso de pruebas y los pasos de identificación, reporte, análisis y verificación de bugs.
- Investigar sobre los siguientes conceptos:
  1. Batch Rendering.
  2. Frustum Culling.
  3. Occlusion Culling.

2. Proceso de Pruebas en Video Juegos (Game Testing)

El proceso de pruebas es parte del desarrollo de todo vídeo juego. Los vídeo juegos son piezas de software de gran complejidad y como toda pieza de software estos pueden poseer cientos de bugs el proceso de pruebas consiste en identificar, reportar, analizar y

verificar distintos bugs este proceso es realizado por distintos grupos de prueba con variados conocimientos de las mecánicas internas del juego durante y al final del desarrollo del juego. En esta sección se explican distintos enfoques para realizar el proceso de pruebas.

2.1. Unit Texting / White Box Testing (Pruebas por Unidades / Pruebas de Caja Blanca)

Esta es la forma mas sencilla para realizar pruebas sobre cualquier pieza de software. Consiste en que una vez finalizado el software, se debe escribir una serie de pruebas en distintas situaciones del estado del programa presentando todo un rango de valores de entradas validos y no validos para luego escribir por cada caso cual produce algún resultado inesperado o correcto [1, p. 155]. Esta clase de pruebas son usualmente realizadas por personas con conocimiento interno de las mecánicas del juego e incluso manejo del código fuente.

2.2. Black Box Testing (Pruebas Caja Negra)

Este tipo de proceso de prueba es comúnmente usado en los últimos pasos del proceso de desarrollo del vídeo juego o incluso cuando este ya se considera ẗerminado; se realizan sobre un grupo pequeño de personas y estas no deben tener conocimiento alguno del código fuente o la lógica interna de todas las mecánicas del juego. Consiste en simplemente presentar la pieza de software a un grupo de prueba para que estos prueben el programa y reporten algún problema que hayan encontrado durante su experiencia. Es común presentar al grupo de pruebas una lista de tareas que hacer durante el juego y un formato estricto de reporte para los problemas, inconvenientes o bugs que estos encuentren.

2.3. Beta Testing (Pruebas Beta)

Beta Testing es muy parecido a Black Box Testing pero estas pruebas son realizadas sobre personas ya interesadas en el juego y sobre un grupo mucho mayor y no existe ninguna lista tareas para el grupo seleccionado. Los desarrollados deben proveer un sistema de reporte de bugs estricto e intentar corregir todos los bugs encontrados durante el proceso de pruebas. El proceso de pruebas beta puede ser cerrado (closed beta) o abierto (open beta),

la diferencia consiste en que mientras open beta es abierto a cualquier persona que quiera participar closed beta esta limitado a un numero especifico de personas.

3. Proceso de Optimizaciones

Como toda pieza de software los vídeo juegos también están propensos a sufrir de bajo rendimiento en algunos de sus componentes e incluso todo el software en total. El proceso de optimizacion a nivel interno es totalmente dependiente del framwork o motor utilizado, muchos motores como por ejemplo Unity3D ya poseen algoritmos de optimización comunes como occlusion culling [8], frustum culling [5] y batch rendering [6] mientras que en otros estos algoritmos deben ser implementados.

Sin embargo existen algunas técnicas comunes que pueden ayudar a mejorar el rendimiento de cualquier juego, están son explicadas en esta sección.

3.1. Texturas Atlas

Una textura atlas es una imagen que contiene variadas sub-imagenes donde cada una de estas sub-imagenes es la textura de alguna parte de un modelo 2D o 3D. Estas sub-texturas son renderizadas modificando las coordenadas de texturas en el mapa uv del objeto en la textura atlas, en otras palabras diciendole en que parte de la imagen se encuentra la textura del objeto. Esta tecnica es particularmente eficiente para casos donde se usan muchas pequeñas texturas que son usadas constantemente, esto reduce de manera significante cambiar el estado del render (render state) cada vez que se tiene que asociar una nueva textura ya que ahora se hace una sola vez, la desventaja es que ahora existe la posibilidad de tener texturas (guardadas en la textura atlas) que tal vez no estemos utilizando alojadas en memoria de GPU.

3.2. Optimización de Mallados

Muchos modelos utilizadas en video juegos en su estado original posee una complejidad geometrica que puede ser excesiva. La optimizacion de mallados busca reducir la cantidad de poligonos en un mallado preservando la calidad del mallado original. Existe

una variedad de software, frameworks y motores de juego gratis o pagos que hacen esto, algunos populares son Simplygon [4] y Polygon Cruncher [3].

3.3. Niveles de detalle

No tiene sentido mantener un modelo de extrema calidad cuando este es apenas visible en pantalla, para evitar esto se usan niveles de detalles. Consiste en tener a partir del mayado original (máximo detalle) un numero n de niveles de detalles e intercambiar el mallado del objeto por el mallado correspondiente a el nivel de detalle del objeto según algún factor como por ejemplo distancia del objeto contra la cámara principal. Algunos motores de juego como Unity3D [7] o jMonkeyEngine [2] ya realizan esto internamente en otros esto debe ser programado manualmente, para crear los distintos modelos de cada nivel de detalle se utilizan los mismos frameworks expuestos en *Optimización de Mallados*.

3.4. Lightmapping

No tiene sentido utilizar iluminación dinámica en una escena estática para evitar esto podemos utilizar lightmaps, un light map no es mas que una textura que almacena la iluminación de un objeto. Los lightmaps puede ser generados previamente y almacenados en imágenes para facil acceso o generados al cargar una escena (calculando la iluminación una sola vez en un frame) y almacenados en una textura para ser aplicada inmediatamente. Una ventaja de los lightmaps es que al estos no ser generados por cada frame sino una sola vez entonces podemos incluir efectos de gran complejidad como iluminación global que serian muy lentos de computar en tiempo real.

3.5. Compresión de Texturas y Mipmaps

Utilizar texturas comprimidas reduce el tamaño de las texturas resultando en tiempos de carga mucho mas rápidos y un menor uso de la memoria, el uso de texturas mas pequeñas puede incluso aumentar la rendimiento del renderizado en general.

Una mipmap es una secuencia de texturas pre-calculadas y optimizadas donde cada textura es una representación de la imagen original con una resolución progresivamente

menor. Todo motor de juego, framework o biblioteca debe proveer una opción para activar mipmaps en una textura.

4. Actividad

Por hacer.

Referencias

[1] Bethke, E. *Game Development and Production*. Wordware Game Developer's Library, 2003.

[2] JMonkeyEngine. Level of Detail (LOD) Optimization. `http://wiki.jmonkeyengine.org/doku.php/jme3:advanced:level_of_detail`.

[3] MooTools Software. Polygon Cruncher SDK. `http://www.mootools.com/plugins/us/3dlibrary/`.

[4] MooTools Software. Simplygon: Automagic 3D Optimization. `https://www.simplygon.com/`.

[5] Unity Technologies. Unity Manual: Camera. `http://docs.unity3d.com/Manual/class-Camera.html`, 2014.

[6] Unity Technologies. Unity Manual: Draw Call Batching. `http://docs.unity3d.com/Manual/DrawCallBatching.html`, 2014.

[7] Unity Technologies. Unity Manual: Level of Detail. `http://docs.unity3d.com/Manual/LevelOfDetail.html`, 2014.

[8] Unity Technologies. Unity Manual: Occlusion Culling. `http://docs.unity3d.com/Manual/OcclusionCulling.html`, 2014.