Computación Gráfica

Trabajo Práctico 1 - Ray Tracing

ITBA 2015

Fecha de entrega: 25/5/2015

Objetivo

El objetivo de este trabajo práctico es el desarrollo de un motor de *Ray Tracing* que permita representar escenas con un balance adecuado entre velocidad y calidad gráfica.

Requerimientos

Features requeridos:

- Cámara pinhole: se debe poder especificar el field of view
- Antialiasing
- Luces: direccionales, ambientales y puntuales.
- Sombras simples
- Refracción
- Reflejos
- Texturas
- Lambert y Phong shading

Todo objeto (luz, cámara, primitiva, etc) deberá poseer una posición, orientación y escala (cuando tenga sentido) arbitraria, definida por una matriz de transformación.

Elementos que debe soportar:

- Esferas
- Planos
- Cajas
- Triángulos
- Meshes

Debe utilizar algún método de subdivisión espacial, ya sea:

- Octree
- KD-Tree
- BSP
- BVH

Características adicionales:

- El renderer debe ser multihilo
- Cronometrado del render
- Se debe poder especificar el *ray depth* de reflejos y refracciones.
- Opciones de antialiasing

Características adicionales opcionales:

- Canales texturizables (diffuse, specular)
- Material mix
- Spot lights

Detalles de implementación

Lenguaje y librerías a usar:
Deberá ser desarrollado en Java, y se permitirán el uso de:

- Toda la biblioteca estándar de Java.
- Librerías auxiliares para operaciones con vectores y matrices (ej: vecmath).
- Librerías para el manejo de imágenes (lectura y escritura, en caso de necesitar algo adicional a ImageIO)
- Librerías de manejo de argumentos de línea de comando como commons-cli de Apache.

NO podrán emplearse librerías externas para las pruebas de colisiones. Consultar a la cátedra ante cualquier duda, antes de usar una librería no permitida.

Formatos de Archivos Gráficos
Los formatos de salida de la imagen deberán ser sin pérdida de datos (lossless), y se deberá implementar por lo menos PNG. Podrán implementarse formatos adicionales (ej:
TGA, BMP).

Se deberán aceptarán como formatos para texturas todos los siguientes: PNG, GIF, BMP, JPG. Tener en cuenta que las escenas deben poder verse con la implementación de referencia.

Formato de escena e implementación de referencia

El formato de escena será un subconjunto del formato de escena de LuxRender (http://www.luxrender.net), con algunas modificaciones. El formato de escena está definido en http://www.luxrender.net/wiki/Scene_file_format_1.0

Para desarrollar, puede utilizarse Blender con el plugin de Lux que puede descargarse. http://www.luxrender.net/wiki/LuxBlend_2.5_installation_instructions

Cualquier función no requerida deberá ser ignorada por el motor en forma transparente, continuando con el render de lo que pueda interpretar.

Se deberán implementar todos los identificadores necesarios para renderizar lo requerido. Sin embargo, tendrán que tenerse en cuenta las siguientes consideraciones:

Materiales:

- **Matte:** ignorar sigma
- **Mirror:** ignorar thin film
- **Glass:** ignorar cauchy, thin film
- **Metal2:** aproximar utilizando Phong. Utilizar roughness como el exponente. Ignorar fresnel.

Camara: implementar “perspective” y field of view, y **LookAt**

Film: implementar xresolution, yresolution

Geometría:

- **sphere:** implementar sólo radius
- **mesh:** Ignorar quads
- **plane:** Si bien no existe en Lux, definir simplemente normal como float3

  Ejemplo:

  Shape "plane"
  "normal N" [0.0 -1.0 0.0]
  "string name" ["Plano1"]

- **box**: Si bien no existe en Lux, definir **width, height, depth** como float

  Ejemplo:

  Shape "box"
  "float width" [10.0]
  "float height" [2.0]
  "float depth" [5.0]
  "string name" ["Cubo1"]

Luces: distant, infinite, point.

Texturas: imagemap

Incluir el parseo de archivos .lxm y .lxo para materiales y geometría. Interpretar la geometría en el formato nativo (identificador mesh) de Lux (en el exportador de Blender, desactivar .ply export)

NO se podrá usar el código fuente de LuxRender para ninguna parte del trabajo. Se permitirá únicamente basarse en el parser existente para implementar uno propio.

Se deberán entregar escenas que muestren TODAS las funciones / características implementadas por el motor (una escena por cada una).

Ejecución y opciones de línea de comando

El programa deberá poder correr en máquinas virtuales de Sun Java.

Deberán entregarse tanto el código fuente como los binarios. El binario generado deberá ser un jar ejecutable, que no requiera mayores dependencias ni la configuración del classpath. Es decir, se debe poder ejecutar via `java -jar xxxx.jar`. Se recomienda usar el plugin `shade` de Maven.

El programa deberá recibir opciones en la línea de comandos de la forma **-<opción> [parámetro]** , o sea, guión seguido del indicador de opción, y de haber un parámetro deberá estar separado por un espacio. El en caso de que detecte una opción desconocida, deberá ignorarla y procesar el resto de las opciones.

Las opciones indicadas como “Opcional” podrán no estar presentes en toda línea de comando, pero **SI** deberán ser implementadas por el programa.

Se deberán implementar las siguientes opciones:

-o <nombre de archivo>
(Opcional) Nombre del archivo de salida, incluyendo su extensión. En caso de no indicarlo usará el nombre del archivo de input reemplazando la extensión y usando el formato PNG.

-i <nombre de archivo>
Nombre del archivo de entrada (definición de la escena)

-time
(Opcional) Mostrará el tiempo empleado en el render (en caso de que no se haya seleccionado, que de todas formas requiere que se muestre el tiempo total y promedio).

-aa <cantidad de samples de antialiasing>
Cantidad de muestras de antialiasing.

-benchmark <n>
(Opcional) Realizar el render completo *n* veces consecutivas.

-d <n>
(Opcional) Define el *ray depth* de reflejos y refracciones.

Entrega digital

Para la entrega, el repositorio debe contener:

- Todos los fuentes correspondientes al proyecto.
- Un README.md que indique como generar los binarios desde línea de comandos.
- Una copia del informe entregado.
- Los binarios ya generados del proyecto.
- Un directorio con todas las escenas de referencia utilizadas por el equipo.

Informe

El informe deberá contener:

- Carátula con número de grupo, nombre de los integrantes y título del trabajo práctico.
- Aclaración de qué método se implementó para los casos opcionales (volúmenes envolventes, subdivisión espacial, etc.) y por qué se implementó ese y no otro.
- Aclaración de qué opcionales se implementaron y por qué.
- Imágenes que muestren las distintas características implementadas.

- Descripción los problemas encontrados durante el desarrollo, y la solución implementada.
- Tabla con tiempos de rendering de distintas escenas (por lo menos las mostradas en el informe), aclarando el hardware usado. En caso de implementar multithreading, comparar casos con distintas estrategias de paralelización implementadas.
- Referencias a las fuentes consultadas para realizar el trabajo.

NO deberá contener:

- Una descripción de qué es un raytracer, de cómo funciona el algoritmo de raytracing, su historia, u otra información de fondo a menos que sea necesaria para explicar en particular algo relevante al trabajo implementado.
- Exceso de imágenes: mostrar las características con la menor cantidad posible de imágenes, dejar adicionales en el directorio “images” del repositorio.

Forma de entrega

La entrega se hará en forma digital por medio de un correo electrónico a la lista de correos del grupo indicando el hash del commit a ser considerado.

Penalidades

- Cualquier entrega con una demora de hasta 30 minutos tendrá una penalidad de 0.5 puntos.
- Entregas con hasta una semana de demora serán penalizados con 2 puntos.
- Entregas con demoras superiores a una semana, se considerarán reprobadas.

Adicionalmente, cualquier entrega que no cumpla con todos los requerimientos establecidos sufrirá una pena adicional de 1 punto.

Evaluación

Para la evaluación del trabajo práctico se tendrá en cuenta:

1. correcto funcionamiento de todos los items obligatorios
2. calidad del informe escrito
3. calidad de las imágenes generadas
4. adicionales implementados por el equipo
5. penalidades

Adicionalmente, se dará 1 punto extra a todo trabajo praćtico que estando en condiciones de aprobar demuestre una performance superior a 3 FPS consistentemente con escenas de 65000 poligonos y 2 luces puntuales, a una resolución de 640x480 en un quad-core de 1.7GHz.

Valen las siguientes aclaraciones:

• Los puntos de implementación corresponden a la implementación completa de todas las características pedidas, y podrán restarse puntos por la no implementación de alguno, o por implementaciones con problemas de ejecución (bugs).
• El informe es obligatorio y los trabajos que no lo entreguen no estarán aprobados.
• Los parámetros subjetivos (calidad) se evaluarán a criterio de los docentes y tomando como referencia los trabajos de otros grupos y de otros años, y se colocarán los puntos que se consideren adecuados hasta el máximo indicado.
• Se sumarán puntos a la nota hasta un máximo de 10 en total. Ej: 11 puntos equivale a un 10. No se acumulan puntos para el trabajo siguiente.

Sugerencias

- Ante una pantalla negra, probar las escenas con la implementación de referencia. Dibujarlas en papel para entenderlas mejor.
- Implementar de a un feature a la vez y luego hacer testing regresivo sobre los anteriormente implementados.
- No buscar performance inmediatamente: experimentar con escenas con pocos objetos para probar cada feature nuevo.
- Encontrar un análogo a **printf** para debuggear: por ejemplo, pintar pixels según ciertas condiciones, etc.
- Aprovechar las facilidades de Java para concurrencia para el caso de multithreaded. En java 8 investigar `ForkJoinTask`
- Ante cualquier duda de interpretación del enunciado, consultar por email a la cátedra.