TPE Nº 2 – Ray Tracing

Fecha de entrega: Miércoles 12 de Mayo al comenzar la clase.

Objetivo

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

Características a implementar

El programa deberá implementar las siguientes funciones:

- Cámara móvil: Deberá poder indicarse una ubicación y orientación arbitraria para la cámara.
- Antialiasing: Podrá ser estocástico o adaptable.
- Múltiples Luces: Para cada una deberá poder indicarse su color y alcance (que podrá ser infinito). El modelo empleado para el *falloff* (lineal, exponencial, etc.) queda librado a la implementación. Deberán implementarse por lo menos luces puntuales (iluminan en todas direcciones). Podrán implementarse otros tipos (spot, etc.).
- Sombras.
- Refracción.
- Reflexión.
- Texturas.
- Phong Shading.

Las escenas deberán permitir elementos de los siguientes tipos:

- Esferas, triángulos, planos y cajas (pueden estar conformadas por triángulos).

Se deberá implementar algún método de volúmenes envolventes. Podrá ser uno de los siguientes u otro (consultar a los docentes antes de elegir otro):

- Cajas.
- AABB.
- Esferas.

Se deberá implementar algún método de subdivisión espacial, que podrá ser uno de los siguientes u otro (consultar a los docentes antes de elegir otro):

- Octrees
- Subdivisión regular del espacio
- KD-Trees
- BSP

Características adicionales a implementar:

- Cronometrado del rendering.

Adicionales no obligatorios: Los siguientes adicionales no son obligatorios, pero sumarán puntos a los grupos que los implementen:

- Implementación multithreaded que aproveche los procesadores multi-core para distribuir el trabajo y disminuir el tiempo de rendering. Se espera un aumento de al menos un 50% con 2 threads por sobre 1 thread para considerar la implementación como funcional (0,5 puntos).
- Profundidad de campo (1 punto).
- Soft-shadows (1 punto).

Detalles de Implementación

Lenguaje y librerías a usar:

Deberá ser desarrollado en Java, y se permitirán el uso de:

1) Toda la biblioteca estándar de Java.
2) Librerías auxiliares para operaciones con vectores y matrices (ej: vecmath).
3) Librerías para el manejo de imágenes (lectura y escritura, en caso de necesitar algo adicional a ImageIO).

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 y BMP. 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 la Escena e Implementación de Referencia

Se usará el formato de escena del programa Sunflow (http://sunflow.sourceforge.net).
Para documentación del formato ver los ejemplos disponibles online y el wiki (http://sfwiki.geneome.net/):

- En el caso de escenas que implementen funciones no requeridas (ej: objetos procedurales), el motor deberá simplemente ignorarlos y continuar con el rendering de los objetos que si pueda interpretar.
- El tamaño de la imagen a generar y otros parámetros, como el antialiasing, se definen en la escena misma, dentro del elemento image.

- La camara a implementar será de tipo **pinhole** o **thinless** (para el opcional de profundidad de campo)
- Para la paralelización del rendering se puede usar el parámetro bucket para indicar el tamaño de cada porción de trabajo.
- Se deberán implementar los shaders **phong con texturas, glass, mirror.**
- Se deberán implementar las transformaciones: **rotatex, rotatey, rotatex, scalex, scaley, scalez, scaleu.**
- Se deberán implementar las primitivas: sp**here, object / generic-mesh, box, plane**
- El modelo de color a usar será "**sRGB linear**"

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

Ejecución y opciones de línea de comando

El programa deberá poder correr en máquinas virtuales de Sun Java 5 o 6.

Deberán entregarse tanto el código fuente como los binarios. Deberá acompañarse de un archivo de ejecución de comandos (*.cmd* en windows, *.sh* en Linux) que permita ejecutarlo incluyendo todo lo necesario para el *classpath* y aceptando opciones de línea de comando.

Se permitirá el uso de ant para administrar la compilación y generar el ejecutable final. Consultar a la cátedra si desean usar otro mecanismo o librería.

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:

`-progress`
(Opcional) El programa deberá mostrar el progreso en el *rendering* mediante una salida periódica en pantalla (ej: impresión de un carácter cada x líneas, o de un porcentaje de completado). En caso de no especificarla el comportamiento *default* será no mostrar nada.

`-gui`
(Opcional) En el caso de que el programa implemente una interfaz gráfica en la que se puedan especificar los parámetros de ejecución, esta opción permitirá mostrarla e ignorar todas las otras opciones de la línea de comandos.

`-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.

Ejemplo: si se especifica `-i casa.sc`, la salida será al archivo `casa.png`.

`-i <nombre de archivo>`
Nombre del archivo de entrada (definición de escena en Sunflow)

`-dof <T>`
(Opcional, solo para los casos donde se implemente) Usar profundidad de campo. El parámetro <T> indica el tamaño (en medidas de la escena) del lente desde el que se proyectan los rayos para simular profundidad de campo.

`-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).

`-show`
(Opcional) Al finalizar el raytracing mostrará una ventana con la imagen. En caso de no especificarlo, el comportamiento *default* será no mostrar la ventana.

Material a Entregar en el SVN

En el directorio TPE02 del SVN correspondiente, colocar:

- Códigos fuente (.java, packages), en el subdirectorio "src".
- Código binario (.class), en el subdirectorio "bin".
- El programa ya compilado, en el subdirectorio "build".
- Un archivo de ejecución de comandos llamado run.cmd o run.sh permita ejecutarlo incluyendo todo lo necesario para el classpath y aceptando opciones de línea de comando.
- Imágenes obtenidas con ejecuciones de prueba en el subdirectorio "images".
- Documento del informe, en el subdirectorio "doc".

En caso de usar Maven u otra herramienta que requiera una estructura de directorios distinta, notificar a los docentes al momento de entregar.

En caso de que el tamaño total de la entrega exceda los 50Mb, notificar a los docentes antes de entregar.

Informe

El informe deberá contener:

- Número de grupo, nombre de los integrantes, título del trabajo práctico, fecha de entrega, etc.
- Descripción los problemas encontrados durante el desarrollo, y la solución implementada.
- 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.

- 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 cantidad de threads usados.

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".

Evaluación

Se evaluará el trabajo de la siguiente forma:

- Implementación de todas las características pedidas: 6 puntos.
- Calidad del informe: 2 puntos.
- Velocidad de ejecución: 1 punto.
- Calidad visual: 1 punto.
- Adicionales: acorde a cada item.
- Entrega tardía (1 semana después): -2 puntos.

Notas:

- Los puntos de implementación corresponen 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, velocidad) 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

- Comenzar donde termina el trabajo de RayCasting.
- 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.
- Aprovechar las facilidades de Java para concurrencia para el caso de multithreaded.
- Ante cualquier duda de interpretación del enunciado, consultar a la cátedra.