

SonSilentSea, creación de juegos en Blender con Python

Jose Luis Cercos-Pita

November 10, 2013

1 Introducción

- ¿Quién soy yo?
- ¿Qué es SonSilentSea?
- ¿Qué pretendo contar?

2 Estructura

- Estructura general
- Componentes

3 Conclusiones

- Resultado del experimento
- Conclusiones
- Demostración

¿Quién soy yo?

- Ingeniero Superior Naval y Oceánico
- Doctorando del programa de ingeniería aeroespacial
- Investigador en el canal de ensayos de la ETSIN
- Especializado en mecánica de fluidos computacional
- Desarrollador de software libre
 - FreeCAD
 - ocland
 - SonSilentSea
 - ...

¿Qué es SonSilentSea?

- Juego de simulación naval
- Dedicado a Sonsoles Jiménez Caballero
- Software libre
- Desarrollado durante 2 años con C++ y OGRE
- Ahora se desarrolla en Blender con Python



¿Qué pretendo contar?

- Blender vs. OGRE
- **C++ vs. Python**

○ más concretamente...

- Diferencias debidas a la familiaridad con el diseño, el lenguaje y el entorno
- Diferencias debidas a una mejor integración
- Diferencias debidas a tener un framework
- Diferencias debidas a los estándares (ligado con la integración)
- **Diferencias debidas al lenguaje (C++ vs. Python)**

Tratamos de estimar que parte se limita al cambio de lenguaje, pero es difícil de medir: tiempo, esfuerzo, cantidad de información...
"2 años en C++ → 2 meses en Python"

¿Qué pretendo contar?

- Blender vs. OGRE
- **C++ vs. Python**

O más concretamente...

- Diferencias debidas a la familiaridad con el diseño, el lenguaje y el entorno
- Diferencias debidas a una mejor integración
- Diferencias debidas a tener un framework
- Diferencias debidas a los estándares (ligado con la integración)
- **Diferencias debidas al lenguaje (C++ vs. Python)**

Tratamos de estimar que parte se limita al cambio de lenguaje, pero es difícil de medir: tiempo, esfuerzo, cantidad de información...

"2 años en C++ → 2 meses en Python"

Precedentes

Usando algunos ejemplos de algoritmos de ordenación...

Python

- Bubble sort: 24 líneas
- Bitonic sort: 23 líneas
- Counting sort: 11 líneas
- Insertion sort: 8 líneas
- Merge sort: 24 líneas
- Heap sort: 35 líneas
- Radix sort: 38 líneas

C++

- Bubble sort: 33 líneas (1.4 líneas más)
- Bitonic sort: 131 líneas (5.7 líneas más)
- Counting sort: 49 líneas (4.5 líneas más)
- Insertion sort: 9 líneas (1.1 líneas más)
- Merge sort: 58 líneas (2.4 líneas más)
- Heap sort: 53 líneas (1.5 líneas más)
- Radix sort: 90 líneas (2.4 líneas más)

Con ésta pequeña muestra crear un algoritmo en C++ parece requerir 2.7 ± 1.6 líneas de código

Diferencias en cuanto a características

Blender

- 20 FPS en el portátil
- Cámara "isométrica"
- Sólo visión desde fuera del agua
- Geometría del mar simplificada por un plano
- Física más realista
- No existe necesidad de preocuparse por la atmósfera



OGRE

- Inmanejable en el portátil
- Cámara libre
- Completo entorno submarino (Hydrax)
- Compleja geometría del mar
- Física pobre
- Atmósfera realista (SkyX)



Estructura general

Blender

7975 líneas de código

main
Blender

frame listener
Blender logic

input manager
Blender logic

Environment
Water plane

GUI
bgui

Sound
Blender internal engine

Entities
Player/AI control & ship motions

OGRE

39211 líneas de código (-1054 líneas)

main
112+257+303 líneas

frame listener
242+140 líneas

input manager
OIS reading

Environment
PSSM + HydraX + SkyX

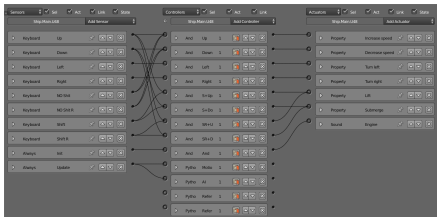
GUI
CEGUI

Sound
ALSA + OGG Vorbis

Entities
Ship motions

Input manager

Blender (0 líneas)



OGRE (-1054-379 líneas)

Analizar las entradas por teclado y ratón en C++ requiere generar una clase específica para ello haciendo uso de OIS (el estándar en OGRE hasta la versión 1.8).

Header file
204 líneas

Source file
175 líneas

Input manager

Blender (-98 líneas)

Camera inputs
83 líneas

Submarine inputs
15 líneas

OGRE (-1054-379 líneas)

Sin implementar en OGRE

Environment (Parallel-Split Shadow Maps)

- Crucial para obtener sombras de calidad.
- Crítico en función de la distancia mínima de corte:

$$\begin{pmatrix} x_v \\ y_v \\ z_v \\ w_v \end{pmatrix} = \begin{pmatrix} \frac{n}{f} & 0 & 0 & 0 \\ 0 & \frac{n}{h} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix},$$

Resultando la profundidad del punto en pantalla como:

$$z_s = \frac{z_v}{w_v} = 2 \frac{fn}{f-n} \frac{w}{z} + \frac{f+n}{f-n} = \mathcal{O} \left(\frac{1}{z} \right)$$

Blender (-98 líneas)

OGRE (-1054-379-159 líneas)

Implementado internamente en el motor de Blender.

Header file
40 líneas

Source file
119 líneas

Environment (Parallel-Split Shadow Maps)

- Crucial para obtener sombras de calidad.
- Crítico en función de la distancia mínima de corte:

$$\begin{pmatrix} x_v \\ y_v \\ z_v \\ w_v \end{pmatrix} = \begin{pmatrix} \frac{n}{f} & 0 & 0 & 0 \\ 0 & \frac{n}{h} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix},$$

Resultando la profundidad del punto en pantalla como:

$$z_s = \frac{z_v}{w_v} = 2 \frac{fn}{f-n} \frac{w}{z} + \frac{f+n}{f-n} = \mathcal{O} \left(\frac{1}{z} \right)$$

Blender (-98 líneas)

OGRE (-1054-379-159 líneas)

Implementado internamente en el motor de Blender.

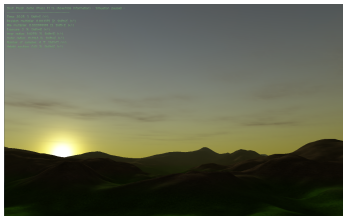
Header file
40 líneas

Source file
119 líneas

Environment (Atmósfera)

Blender (-98 líneas)

La cámara no permite apreciar la atmósfera, y por tanto no ha sido implementada.



OGRE (-1054-379-159-11877 líneas)

SkyX
11877 líneas



Environment (Océano)

Blender (-98-38 líneas)

Plano que sigue a la cámara
38 líneas

Olas creadas internamente en el shader
0 líneas

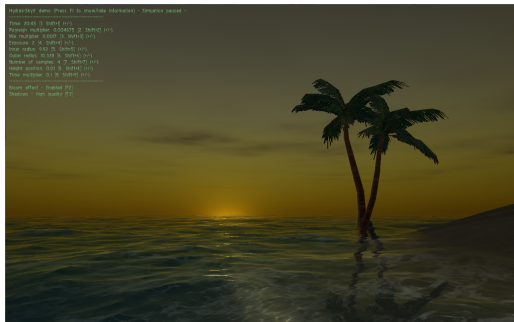
Sin entorno submarino
0 líneas

OGRE (-1054-379-159-11877-8220 líneas)

Geometría compleja
121+159+193+229+277+452+667+842 líneas

Olas generadas sobre la geometría
 $65+97+112+133+135+152+186+194+272+282+589+800$ líneas

Entorno submarino
 ~43+~75+335+~190+~255+915+~450 líneas



GUI

Blender (-98-38-2500 líneas)

bgui tool
2500 líneas

Código para manejar las ventanas
362 líneas (que cuentan para comparar)

OGRE (-1054-379-159-11877-8220 líneas)

CEGUI
0 líneas

Código para manejar las ventanas
1895 líneas (que cuentan para comparar)

Sonidos

Blender

(-98-38-2500 líneas)

Gestor de sonidos interno de Blender
0 líneas

OGRE

(-1054-379-159-11877-8220-1880 líneas)

Gestor de sonidos & streamer
1880 líneas

Entidades

Blender

(-98-38-2500-2206-388 líneas)

Gestor de partículas adaptado de easyEmit
2206 líneas

Control del jugador / inteligencia artificial
138+194+56 líneas

OGRE

(-1054-379-159-11877-8220-1880-592 líneas)

Gestor de Partículas asistido por OGRE
592 líneas

Sin control implementado
0 líneas

Resultado

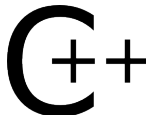
Blender

2745 líneas de código “comparables”



OGRE

15050 líneas de código “comparables”



Según la estimación (en la que he tratado de restringir al máximo los efectos al cambio de lenguaje de programación) el código en C++ parece requerir 5.5 líneas más de código que una implementación en Python.

La medida queda fuera del valor que obtuvimos muestreando algunos algoritmos de ordenación (2.7 ± 1.6).

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Conclusiones

- Existe una gran cantidad de información y comparaciones fiables sobre la velocidad de códigos implementados en ambos lenguajes.
- Existen estimaciones del número de errores que se cometen en cada lenguaje por línea de código escrita (0.69 vs. 0.005 bugs per KLOC).
- No parecen existir estudios serios acerca de la diferencia de esfuerzo en el desarrollo dependiendo del lenguaje empleado. Por ejemplo, si en C++ se requieren 10 líneas más de código que en Python para llevar a cabo la misma tarea, el índice de errores estará artificialmente incrementado un orden de magnitud.
- Se ha tratado de estimar la influencia del lenguaje de programación en una aplicación compleja, usando como indicador el número de líneas de código. Para ello se ha tratado de eliminar todo elemento “no comparable”.
- Ambas aplicaciones son demasiado diferentes, así que el resultado no debe considerarse una buena medida.
- Tal y como se podía esperar en C++ se ha obtenido un incremento significativo de las líneas de código necesarias.
- Tampoco se ha abordado la dependencia con el ámbito de aplicación.

Demostración

Demo