22.08 Algoritmos y Estructuras de Datos

Level 1: Warm Up

Introducción

En esta primera práctica vamos a simular el sistema solar en 3D.

Physics brushup

La fuerza gravitatoria ejercida sobre un cuerpo de masa m_i ubicado en $\mathbf{x}_i$ por un cuerpo de masa m_j ubicado en $\mathbf{x}_j$ es:

$$\mathbf{F}_{i,j} = -\frac{G m_i m_j}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}\hat{\mathbf{x}}_{i,j}$$

$\hat{\mathbf{x}}_{i,j}$ es un vector unitario que va del cuerpo j al cuerpo i:

$$\hat{\mathbf{x}}_{i,j} = \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$$

La constante gravitatoria G es:

$$G = 6.6743 \times 10^{-11}\frac{N \cdot m^2}{kg^2}$$

La fuerza neta $\mathbf{F}_i$ que actúa sobre el cuerpo i es:

$$\mathbf{F}_i = \sum_{i=1, i\neq j}^{N} \mathbf{F}_{i,j}$$

Para determinar la aceleración $\mathbf{a}_i$, aplicamos la segunda ley de Newton:

$$\mathbf{a}_i = \frac{\mathbf{F}_i}{m_i}$$

Para calcular la posición $\mathbf{x}_i$ debemos integrar la aceleración dos veces. Pero debemos hacerlo en forma discreta, ya que nuestras computadoras son digitales.

Las siguientes ecuaciones permiten aproximar las integrales de velocidad y de posición con un **time step** Δt:

$$\mathbf{v}_i(n+1) = \mathbf{v}_i(n) + \mathbf{a}_i(n)\Delta t$$

$$\mathbf{x}_i(n+1) = \mathbf{x}_i(n) + \mathbf{v}_i(n+1)\Delta t$$

Si el error de la aproximación es demasiado grande, debes disminuir Δt. Pero cuidado, ¡disminuirlo aumenta también el cálculo necesario!

Nota: Es importante que actualices la posición $\mathbf{x}_i(n)$ con $\mathbf{v}_i(n+1)$ y no con $\mathbf{v}_i(n)$. Hacerlo con $\mathbf{v}_i(n+1)$ mejora notablemente la precisión.

Organización

Ahora que has adquirido una comprensión del problema, estás listo para escribir código. ¿Pero cómo podemos organizar este proyecto de manera efectiva?

Una buena idea es dividirlo en modelo, vista y controlador, una estrategia que aporta claridad y orden:

- El **modelo** es el núcleo de la simulación. Aquí es donde almacenamos el estado actual de la simulación a través de variables representativas, y lo actualizamos a través de funciones de actualización.
- La **vista** tiene por objetivo representar gráficamente el modelo. Esta capa se ocupa de dar vida visual a los elementos del modelo.
- El **controlador** se enfoca en la interacción con el usuario. Aquí es donde los elementos de interacción, como botones o entradas, se integran para influir en el modelo y en la vista.

Este enfoque es muy utilizado en el diseño de interfaces gráficas y se conoce como MVC (modelo-vista-controlador).

Para implementar el modelo y la vista, parte del *starter code* que te damos. Debes usar la biblioteca gráfica raylib, que puedes instalar usando vcpkg. En este proyecto no necesitas implementar un controlador ya que raylib lo hace por ti.

Modelo

Empieza preguntándote qué variables de estado necesitas.

La aceleración de cada cuerpo depende sólo de las fuerzas instantáneas aplicadas sobre él, pero la actualización de velocidad y posición requiere el estado anterior. Por tanto debes almacenar la **posición** y la **velocidad** de cada cuerpo. Para la

simulación necesitas además la **masa** del cuerpo, y para la representación gráfica, su **tamaño** y **color**. Puedes encapsular todas estas variables en el tipo de datos `OrbitalBody` del archivo `orbitalSim.h`.

Define asimismo un tipo de datos `OrbitalSim` que contenga el valor del **time step** (el tiempo transcurrido desde el comienzo de la simulación), el **número de cuerpos** en la simulación y un **puntero** a un arreglo de `OrbitalBody`s. Esto permite encapsular el estado de la simulación en una sola variable.

Para crear una simulación, completa la función `constructOrbitalSim` que recibe el valor del time step y devuelve un puntero a un nuevo `OrbitalSim`. Aprovecha los datos del archivo `ephemerides.h` que te proveemos en el starter code.

Luego completa la función `updateOrbitalSim` para simular un time step. La función recibe como parámetro un puntero a un `OrbitalSim`.

Realiza la simulación en dos pasos: calcula primero todas las aceleraciones, y actualiza recién entonces las posiciones y velocidades. Si no separas estas operaciones y haces todo en un solo bucle, la aceleración de algunos objetos se calculará a partir de $\mathbf{x}_i(n+1)$ y no de $\mathbf{x}_i(n)$.

Truco: Para evitar otra asignación de memoria, puedes añadir un campo `acceleration` a `OrbitalBody`, que utilizas en `updateOrbitalSim` como variable temporal entre el primer y segundo paso.

Para liberar los recursos de una simulación, completa la función `destroyOrbitalSim`.

Consejos:

- **Aprovecha la biblioteca de vectores raymath que viene con raylib:** Tan solo tienes que escribir `#include "raymath.h"`. Los siguientes tipos y funciones son particularmente útiles: `Vector3`, `Vector3Add`, `Vector3Subtract`, `Vector3Length` y `Vector3Scale`.
- **Optimiza el cálculo:** Para calcular x^2, `(x * x)` requiere muchísimo menos cálculo que `pow(x, 2)`. También puedes ahorrar cálculo evitando multiplicaciones y divisiones inútiles, como la multiplicación y división por m_i en $\mathbf{F}_{i,j}$ y $\mathbf{a}_i$, respectivamente.
- **Verifica condiciones límite:** Asegúrate que tu código no produce divisiones por cero u otras condiciones que puedan romper la simulación.
- **Usa el debugger:** Aprovecha nuestra guía de **debugging**.

Vista

Para representar los cuerpos celestes en 3D, completa el apartado “Fill in your 3D drawing code here:” de la función `renderView` del archivo `View.cpp`. Puedes usar la función `DrawSphere` de raylib. Para que los cuerpos quepan en la pantalla debes escalar los valores de posición de `OrbitalSim`: recomendamos el factor de escala `1E-11`. Para que las esferas sean visibles también te sugerimos escalar sus radios: La ecuación empírica `0.005F * logf(radius)` da buenos resultados. Si te alejas mucho, verás que las esferas dejan de ser visibles. Para evitar esto, píntalas también con `DrawPoint3D`.

Para mostrar información adicional sobre la simulación, completa el apartado “Fill in your 2D drawing code here:” de `renderView`. Es particularmente útil que llames a `DrawFPS` para mostrar la cantidad de fotogramas por segundo. También puedes mostrar el tiempo de simulación con `DrawText`. Para convertir el tiempo en una fecha en formato ISO puedes usar la función `getISODate` que te proveemos.

Trucos:

- **Usa el teclado y mouse** para moverte a lo largo de la simulación: usa las teclas WASD para moverte adelante, a izquierda, atrás y a derecha, y Q y E para rotar a izquierda y a derecha.

Pasos siguientes

¡Felicitaciones, hiciste tu propio Universe Sandbox! Pero aún quedan cosas por hacer:

1. Verifica que el timestep sea adecuado. Escribe tu justificación en `README.md`. Consejo: haz cambios y fíjate si después de cierto tiempo llegas al mismo resultado.

2. Es probable que hayas usado `float` para representar las masas, distancias, velocidades y aceleraciones de la simulación. Verifica que el tipo de datos que elegiste tenga la precisión y rango necesarios para la simulación. Escribe tu justificación en `README.md`. Consejo: fíjate en el valor máximo que tu simulación necesita, y en cómo la cantidad de decimales afectan al resultado.

3. Añade 500 asteroides modificando `constructOrbitalSim`. Aprovecha la función `configureAsteroid` que te damos. Asegúrate de completarla.

4. Habrás notado que la simulación del punto anterior se volvió terriblemente lenta. ¿Cuál es la complejidad algorítmica de la simulación? ¿Por qué? Escribe tu respuesta en `README.md`.

5. Soluciona el problema anterior. Cuando hayas solucionado el problema, deberías poder simular sin problema 1000 asteroides o más. Pista: hay dos cuellos de botella: uno que involucra a los gráficos, y otro, a la complejidad computacional. Describe los cambios que hiciste en `README.md`.

Bonus points

Cosas optativas para hacer:

- Averigua qué pasaría si Jupiter fuera 1000 veces más masivo, o si un agujero negro pasara por en medio del sistema solar. Aprecia el hecho que estés con vida.
- Las efemérides astronómicas son tablas que describen la posición y velocidad de los cuerpos celestes en un momento dado. En `ephemerides.h` hay efemérides para el sistema Alpha Centauri, la segunda y tercera estrella más cercana al sistema solar. ¡Pruébalas en tu simulador!
- Encuentra el easter egg.