Tarea 3

Instrucciones Generales

- La tarea es estrictamente individual.
- Este enunciado presenta varias opciones, escoja e implemente solo una de ellas.
- El plazo de entrega se indica en tareas/u-cursos.
- DEBE utilizar: Python 3.7 (o superior), Numpy, OpenGL core profile, GLFW.
- *Las imágenes presentadas son solo referenciales, utilice un estilo propio para su trabajo.*

Entregables

- Código que implemente su solución (4.5 puntos)
  - Su versión final DEBE utilizar archivos *.py, NO jupyter notebooks.
  - Incluya TODO lo que su código necesita, incluyendo archivos proporcionados en cátedras o auxiliares.
- Reporte de documentación de entre 1 y 2 planas. Formato pdf. Detalles disponibles en primera clase. (1.2 puntos)
- Vídeo demostrativo de 20-30 segundos. (0.3 puntos)

Objetivos

- Desarrollar una aplicación gráfica que simula problemas físicos (3A), biológicos (3B) o químicos (3C).
- Programar algoritmos simples de detección de colisiones.
- Aplicar métodos de diferencias finitas para EDPs.
- Aplicar y visualizar usando modelo numéricos.
- Visualizar los resultados de la simulación usando OpenGL.
- Trabajar con interacciones de usuario vía GLFW.
- Hacer uso del patrón de diseño Modelo-Vista-Controlador.
- *Nota: No todas las opciones de tareas ejercitan TODOS los objetivos.*

Opción A: Efecto Bloom

El efecto bloom se utiliza para representar de manera relativamente realista y barata la difusión de la luz. Cuando solo se tiene la imagen de una parte del escenario, esta es una aproximación decente de la luz en la escena y para lograr ejecutar el "Texture Baking", que es la preparación de texturas para no tener que calcular la luz en tiempo real.

Se observa en la imagen que los píxeles claros, de donde sale la luz, que esta se difumina, volviendo los colores cercanos al agujero más claros. Para esta tarea, utilizaremos EDPs para simular la difusión de la luz. La ecuación que sigue la luz es la siguiente:

$$\Delta f = 0$$

En este caso, el programa recibe la siguiente llamada:

```
python bloom_effect.py image_filename N R G B
```

Donde `image_filename` es el nombre de la imagen y N es la cantidad de píxeles de alcance que tiene la difusión y los últimos 3 valores son los componentes RGB del color que se difumina a los alrededores (que van de 0 a 255 cada uno). La salida se guarda en un archivo con nombre `image_filename_out`.

Considere lo siguiente:

- Su programa debe recibir una imagen, detectar los puntos donde el color sea el indicado en la llamada y difuminarlos hasta N píxeles de distancia.
- Considere que en este enunciado se indican números enteros para indicar los colores, porque es lo que se usa genéricamente en las imágenes y cámaras. Sin embargo, en los códigos se representa con un valor float de 0 a 1. Puede hacer la conversión fácilmente dividiendo el color dado por 255.

- Un ejemplo más concreto. Supongamos que tenemos una imagen de 20x20 llamada bloom.png. Esta tiene un punto brillante con RGB = (255, 255, 254) en el píxel (10, 10), justo al centro. Si se extiende un N = 2, entonces se verán modificados los siguientes píxeles vecinos: a la izquierda: [9, 10], [8, 10];
  a la derecha: [11, 10], [12, 10];
  arriba: [10, 11], [10, 12];
  y abajo: [10, 9], [10, 8]

  Debe considerar también los casos diagonales, para estas distancias puede hacer una aproximación de la norma euclidiana, o bien usar una norma 1. En ambos casos, para el ejemplo los píxeles diagonales son:
  arriba izquierda: [9, 11]
  arriba derecha: [11, 11]
  abajo izquierda : [9, 9]
  abajo derecha : [11, 9]

  La llamada a ejecutar sería:

  ```
  python bloom_effect.py bloom.png 2 255 255 254
  ```

- Para formar el sistema, piense que una condición de borde no necesariamente es el fin/borde/extremo del dominio. En este caso, la condición de borde conocida son todos los puntos de la imagen donde los píxeles tienen el color indicado en la llamada. Es más ¡En el ejemplo anterior la condición de borde está al medio!

- Como se trata de una imagen, debe preocuparse por extender la luz a lo largo de dos dimensiones. Por otra parte, considere que tanto el color rojo, como verde y azul se vuelven más claros.

- Al formar este sistema, las incógnitas representarán aquellos píxeles en una N-vecindad del color de la luz que se difumina. Sin embargo, **no olvide que estos píxeles ya tenían un color original, el cual no se considera al resolver sistema, pero sí en el resultado.** Esto es, resolver el sistema le dará el brillo de un píxel contiguo, el cual tiene un color original, al cual debe sumarse el brillo obtenido. Por ejemplo, si el brillo obtenido por al resolver el sistema de ecuaciones es [10, 10, 10] para un píxel[i, j], el cual tenía un color rojizo al inicio [210, 40, 30], entonces pasará a ser [220, 50, 40].

- Utilice matrices sparse para resolver su sistema de ecuaciones, de lo contrario se puede acabar la memoria de su programa. Por esto mismo conviene resolver el problema para los colores por separado.

- Puede resolver el sistema de forma iterativa o con el método de Gauss, invirtiendo una matriz.

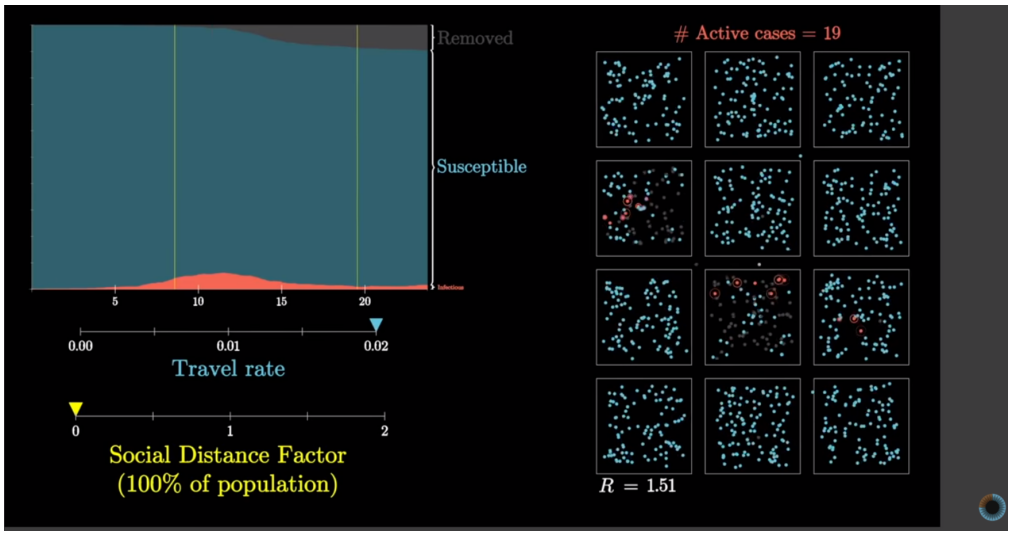
- Elija según su criterio el valor de h. Pruebe para distintos valores de h, distancia entre puntos. Como se trata de píxeles, la distancia es fija, no se recomienda realizar ampliaciones ni disminuciones de tamaño en la imagen original.

Puntuación:

- El programa atiende cualquier imagen, recibiéndola en su llamada: 0.4 pts.
- Se detectan los puntos de interés: 0.4 pts.
- Se resuelve un sistema de EDPs: 1.0 pts.
- Se considera un N variable. Si no lo toma, es probable que el programa se demore una infinidad y consuma muchísima memoria, si es que no se cae: 0.5 pts.
- Se suma el brillo en las tres dimensiones de colores RGB: 0.2 pts.
- Se consideran todos los casos posibles con respecto a condiciones de borde: 0.5 pts.
- Se utilizan matrices sparse para resolver el sistema: 0.5 pts.
- El informe debe ser más extenso para esta tarea, procure explicar los siguientes aspectos: Qué casos importantes hay (píxeles por abajo, arriba de la luz), cómo se resuelve para cada color distinto (no es necesario ahondar en el método matemático, eso se ve en clases. Explique cómo se aplicó ese método a su problema en específico): 0.5 pts extra además del informe, por lo que el informe vale 1.7 pts en este caso.
- No cumplir con el formato requerido sin previo aviso ni justificación es causal de descuento de hasta un punto. Por ejemplo, no utilizar el nombre bloom_effect.py, no entregar un .py o invertir el orden de los argumentos de la llamada. También se descuenta por entregar versiones no funcionales. En casos de discrepancia, su vídeo constituye un respaldo de que su programa funcionaba en su computador.

Opción B: Simulación de contagios

Un nuevo virus está barriendo con la humanidad. Una gran mayoría ya perdió la esperanza, mientras que hay quienes dedicaron sus corazones a procurar un futuro para las futuras generaciones. ¡La OMS necesita nuestra ayuda de manera urgente!



Contemple las visualizaciones en el siguiente vídeo para apreciar las maravillas de la computación gráfica unida a la interpretación de datos: 3Blue1Brown - Simulating an epidemic.

Su programa será llamado de la forma:

```
python pandemic_simulator.py virus.json
```

Un ejemplo de archivo virus.json:

```
[
    {
        "Radius": 0.1,
        "Contagious_prob": 0.2,
        "Death_rate": 0.1,
        "Initial_population": 1000,
        "Days_to_heal": 5,
    }
]
```

Donde:

- **Radius** es el radio de contagio entre una persona y otra.

- **Contagious_prob** es la probabilidad de contagio. Esto significa que cuando una persona sana se encuentra en un radio R que rodea a alguien infectado, se va a contagiar con esa probabilidad.
- **Death_rate** es la probabilidad de muerte por contagio.
- **Initial_population** es la población inicial.
- **Days_to_heal** indica la cantidad de días necesarios para recuperarse.

Puede suponer que siempre habrá una única persona infectada de manera aleatoria al inicio.

Considere los siguientes requisitos:

- Se pide una visualización para mostrar una población donde se pueda observar las posiciones de cada persona junto con su estado de salud. Esto puede ser en 2D ó en 3D. Si su tarea es en 2D se le exigirán más puntos explicados posteriormente.
- El programa avanza por iteraciones que representan días. Se avanza una iteración presionando la tecla derecha. No es necesario implementar el retroceso.
- Se muestra el avance de cada iteración en la consola (con un número: día n) o en la ventana de OpenGL (pueden ser barras en algún extremo de la ventana). Recuerde que puede aumentar el tamaño del Viewport (glfw window) para mayor comodidad.
- En cada iteración, las personas infectadas que estén en un radio Radius cerca de otra sana tienen una posibilidad Contagious_prob de contagiarlas.
- En cada iteración se desplazan las personas según algún comportamiento que usted debe asignar. Este comportamiento debe tener algo de lógica (por ejemplo, grupos de personas que se mueven juntas, como familias), o bien un desplazamiento aleatorio, con una distancia máxima recorrible por iteración (las personas no se teletransportan).
- Las personas contagiadas están claramente diferencias de las sanas a nivel visual.
- Se lleva una cuenta diaria y total de las personas sanas, enfermas, fallecidas y recuperadas.
- Una persona recuperada no se puede volver a enfermar.
- La visualización se puede cortar con la tecla P. Luego, se despliega un gráfico que representa la cantidad de personas contagiadas, sanas y fallecidas durante el transcurso. Debe mostrar dos versiones de estos gráficos: una usando alguna librería de Python (Por ejemplo matplotlib), y otra con OpenGL.

- Una persona contagiada estará enferma durante Days_to_heal días, cada uno de estos días, tiene una posibilidad Death_rate de desaparecer del mapa. Estas desapariciones también se deben contabilizar.
- n = 1 elemento si es 3D, n = 2 elementos si es 2D:
  Agregue n elementos extra a su tarea que se relacionen con una medida del comportamiento de las personas o bien con la interactividad de su programa. Por ejemplo, puede agregar el comportamiento de familias en las personas, grupos religiosos que se conglomeren cada cierto tiempo en un lugar, un mercado central donde la gente se junta, simular varias poblaciones separadas donde existe una posibilidad pequeña de que alguien viaje a otra. Agregar otro parámetro que determine la cantidad de días antes de poder volver a contagiarse. Una opción de interactividad con el programa es poder cambiar los parámetros de radio de contagio, probabilidad de contagio o de muerte apretando ciertas teclas. El cambiar el color de las personas o hacer transformaciones NO aporta mayor información y no se considerará válido. Estos elementos extra deben ir en el comentario de la tarea al entregarla para facilitar su revisión.
- Considere que esta tarea evalúa la parte relacionada con la visualización e interpretación de datos, además de demostrar un dominio acorde de OpenGL.
- La forma del gráfico queda a su criterio, puede ser en barras, en puntos, etc. Se considera correcto si un tercero es capaz de interpretarlo. Puede publicar fotos de sus resultados en el foro para pedir feedback. Pueden compartirse vídeos o imágenes de sus trabajos sin problema, mientras no se compartan código.
- Las iteraciones discretas representando días no son obligatorias, pero son lo más simple. Puede hacerlo continuo si así lo desea.

Puntuación:

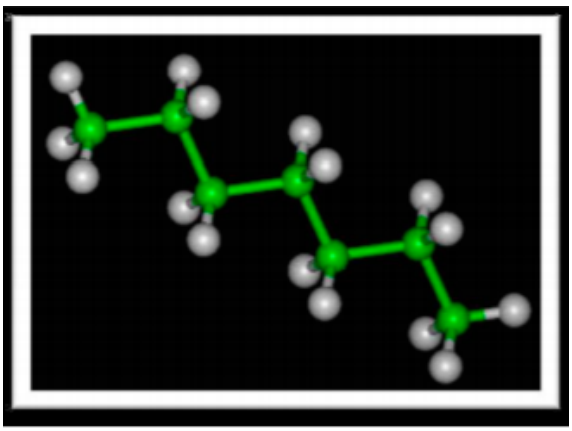
- Considerar el archivo de entrada: 0.5 pts.
- Se observa una evolución del desarrollo del programa (cambios): 0.5 pts.
- Se lleva una cuenta total y actual: 0.2 pts.
- Se visualizan las personas enfermas y contagiadas, en conjunto con sus movimientos entre iteraciones: 1.0 pts.
- Se despliega el gráfico en matplotlib al final: 0.4 pts
- Se despliega un gráfico en OpenGL al final: 0.6 pts.
- Se cumplen los comportamientos esperados, de contagio, recuperación: 0.5 pts.
- Se agregan los elementos extra: 0.8 pts.

Opción C: Hidrocarburos

Los hidrocarburos son compuestos orgánicos formados únicamente por átomos de carbono e hidrógeno. La estructura molecular consiste en un armazón de átomos de carbono a los que se unen los átomos de hidrógeno. Un átomo de carbono puede mantener cuatro enlaces, mientras que el hidrógeno solo puede mantener uno.

Se desea **simular** el comportamiento que tendrían estos átomos al ser depositados en una caja en distintas proporciones, y estudiar cuáles serian las estructuras de hidrocarburos generadas.

En esta ocasión se modelará el problema sin considerar efectos de gravedad, pero considerando las interacciones gravitacionales entre los átomos y moléculas. Esto es, en una caja, se inicialmente se distribuirá uniformemente una cantidad `NCarbono` de átomos de carbono, y una cantidad `Nhidrogeno` de átomos de hidrógeno. Los átomos pueden ser modelados como esferas que rebotarán inelásticamente en los bordes de la caja (en cada colisión la energía se conserva).



En esta tarea, debe utilizar **OpenGL** para dibujar. Como lenguaje base se recomienda utilizar **Python**.

Considere lo siguiente:

- Utilizando OpenGL, dibuje un modelo esférico para el hidrógeno y otro para el oxígeno. Utilice tamaños proporcionales a los radios atómicos (carbono tiene 70 pm de radio medio, mientras que el hidrógeno tiene solo 25 pm). Por otro lado, también debe dibujar una representación para un enlace simple. Considere la malla de cada esfera de tal manera que logre una buen rendimiento en su máquina (ello es, encontrar la discretización polar y azimutal de la esfera para lograr un resultado rápido en ejecución).

- Las partículas se moverán al interior de una caja. Esta caja debe ser modelada de forma que en su interior se logre un buen efecto de iluminación (i.e. utilizar una malla de polígonos relativamente fina); también puede probar diferentes *shaders* para lograr el mismo objetivo de que se vea bien.

- Configure una fuente de luz al interior de la caja. Todos sus modelos deben utilizar sombreado, ya sea FLAT o SMOOTH.

- Al comienzo del programa, se le debe solicitar al usuario que ingrese la cantidad de átomos de hidrógeno y de carbono que se utilizarán en la simulación. Acto seguido, se da inicio a la simulación, donde los átomos son distribuidos uniformemente al interior de la caja. La velocidad inicial para cada partícula es nula. La cámara debe estar inicialmente en una esquina y mirando hacia el interior de la caja.

- La cámara debe poder moverse con las flechas del teclado, rotando hacia los lados y hacia arriba y abajo. Con las teclas "z" y "x" la cámara debe moverse continuamente hacia adelante y atrás.

- Cada átomo debe interactuar con los demás según la ley de gravitación universal, esto es, una fuerza atractiva proporcional al producto de las masas involucradas, e inversamente proporcional al cuadrado de la distancia.

$$\vec{F} = -\frac{Gm_1m_2}{d_{1,2}^2}$$

  Puede despreciar la acción de esta fuerza si los átomos están lo suficientemente distantes. La fuerza neta sobre un átomo estará dada entonces por la suma de las interacciones con cada otro átomo o molécula. A partir de esta fuerza neta, puede calcular la aceleración instantánea de dicha partícula. Para las masas utilice magnitudes proporcionales a las masas atómicas de los isótopos más típicos (1 uma para el hidrógeno, y 12 uma para el carbono).

- Si dos átomos colisionan, de ser posible se formará un nuevo enlace (i.e. ambos con capacidad para un enlace adicional), sino, cada átomo simplemente seguirá su camino (se traslaparán sus trayectorias). En el caso de una colisión carbono-carbono, sólo se considerarán enlaces simples.

- Asuma que los enlaces se distribuyen uniformemente alrededor de cada átomo de carbono central. Esto es, cada átomo de carbono se localizará al centro de un tetraedro, y los otros átomos en los vértices.

- No se complique con la transformación gradual de la molécula, luego de la colisión, modifique la molécula directamente a la forma final. Ignore además las reacciones intramoleculares, es decir, un átomo particular de la molécula no se siente afectado por

otros átomos de la misma molécula. Tampoco se complique con que dos átomos de la misma molécula se sitúen en regiones superpuestas.

- **(Bonus por 0.4pts.)** Considere que dos átomos no pueden superponerse en la misma región. Para ello implemente un sistema de colisiones que verifique que dos átomos no puedan estar en la misma región. Ojo: Si verifica las colisiones con fuerza bruta (evaluar todos con todos) su programa será muy lento; para ello puede aplicar alguna heurística como ir almacenando las distancias de cada átomo en cada iteración y comprobar colisiones con los más cercanos, o escoger de manera aleatoria una muestra de átomos para comprobar colisiones, entre otros.

El programa se ejecutará de la siguiente manera:

```
python hidrocarburos.py Nhidrogeno Ncarbono
```

En donde **Nhidrogeno** y **Ncarbono** es el número entero, mayor a cero, de átomos de hidrógeno y de carbono que se agregan como condición inicial de la simulación.

Puntuación:

- Dibujo modelos hidrocarburos (individual y en enlace): 0.5pts.
- Dibujo caja contenedora de los hidrocarburos: 0.3pts.
- Fuente de luz: 0.2pts.
- Condición inicial del programa, recepción del ínput, ubicación de la cámara y velocidad: 0.3pts.
- Cámara móvil: 0.4pts.
- Simulación, interacción fuerza entre elementos: 1.5pts.
- Simulación, colisión y formación de enlaces (modificación de la estructura geométrica / transformaciones / etc.): 1.3pts.