



Universidad de Chile

Facultad de Ciencias Físicas y Matemáticas

DCC

CC3501-1

Tarea 3: Diferencias Finitas para EDPs y Visualización Científica

Reporte de Documentación tarea 3a

Curso: CC3501-1

Alumno: Sebastián Salinas

Profesor: Daniel Calderón

Auxiliares: Alonso Utreras

Nelson Marambio Q.

Ayudantes: Beatriz Grabolosa M.

Heinich Porro Sufan

Nadia Decar

Tomás Calderón R.

Fecha: 26 de Julio del 2020

1. Descripción del Problema y Arquitectura del Programa

La tarea3a consistía en resolver la ecuación del calor para el agua contenida dentro de un acuario, con el objetivo de poder visualizar en un futuro las distintas zonas de preferencia de los peces dentro del acuario.

Para completar este objetivo se pedía hacer 2 programas, *aquarium_solver.py* y *aquarium_view.py*. Donde en el primero se tenía que resolver la ecuación del calor mediante la resolución del problema de Laplace para la temperatura. Para esto había que discretizar el dominio adecuadamente (en este caso un paralelepípedo) y proceder a resolver la EDP con las condiciones de borde impuestas en el enunciado (las cuales serían recibidas a través de un archivo con extensión *json*).

Y en el segundo, crear una visualización del acuario, ubicando múltiples peces dentro de él, y mostrando con *voxels* las zonas de preferencia de estos. El número de peces y las temperaturas ideales de cada pez también serían especificadas en un archivo *json*, solucionando de esta forma el problema de modelamiento de la temperatura para un acuario y la distribución de los peces dentro de él.

Para implementar el programa *aquarium_solver.py* se siguió una lógica similar a los archivos de ejemplo *ex_finite_differences_laplace_dirichlet.py* y *ex_finite_differences_laplace_neumann.py*.

Sin embargo, como el dominio es en 3D se modificaron las funciones que discretizaban el dominio y lo parametrizaban. En este caso se utilizó un *getIJK(P)*, que es una función que retorna los índices que discretizan el dominio 3D y un *getP(i,j,k)*, que es una función que retorna un P que parametriza esos i,j,k.

Como se mencionó anteriormente las condiciones de borde del problema se recibían a través de un archivo *json*, por lo cual para recibir este input lo que se hizo fue crear el módulo *json_reader.py*, el cual lee un archivo *json* y retorna un diccionario Python con los datos del archivo.

Con la parametrización lista, las condiciones de borde recibidas y un “paso” a elección se prosiguió a ponerse en los 27 diferentes casos que tenía el problema, añadiendo 2 casos extras que corresponden a la regulación de los calentadores de la parte inferior, dando un total de 29 casos. Con todos estos casos resueltos podemos generar un montón de ecuaciones y ordenar las incógnitas en un vector columna, donde podemos usar álgebra estándar para resolver los sistemas de ecuaciones del problema, obteniendo finalmente la resolución de la EDP y guardándola en un archivo *numpy* con el nombre especificado. Cabe destacar que en caso de ser necesario se utilizaron matrices “Sparse” para ahorrar memoria si es que el paso de discretización que se utilizaba era muy pequeño.

Por otro lado, para implementar el programa *aquarium_view.py* se ocupó como base cinco de los módulos presentados por el profesor, estos son *transformations.py*, *basic_shapes.py*, *scene_graph.py*, *easy_shaders.py* y *lighting_shaders.py* y nuevamente el módulo *json_reader.py* para generar el diccionario que contiene las especificaciones de temperatura ideal, número de peces y nombre del archivo con la resolución de la EDP. Además se crearon tres módulos extra, los cuales manejan ciertos aspectos y lógicas que se explicarán a continuación.

El módulo *voxels.py*, que contiene la función *mergeVoxels(...)*, que tiene como propósito generar una sola *gpuShape* de todos los voxels que estén en cierto rango, haciendo que sea mucho más fácil dibujarla. En este caso el rango será el de las temperaturas adecuadas para cada tipo de pez.

El módulo *createFish.py*, el cual contiene una clase *fish()* y 3 funciones: *createClownFish()*, *createSurgeonFish()* y *createShark()*, las cuales se encargan de crear modelos 3D de peces (un pez payaso, un pez cirujano y un tiburón respectivamente) utilizando gráficos de escena.

Por último el módulo *createAquarium.py*, el cual (de manera similar que en el módulo anterior) contiene una clase *aquarium()* y 2 funciones: *createAquariumEdges(x,y,z)* y *createAquariumWindows(x,y,z)*, las cuales se encargan de crear el marco del acuario y de crear las ventanas con las medidas recibidas como parámetro, utilizando nuevamente gráficos de escena. Cabe destacar que las ventanas serán dibujadas con un shader que contenga transparencia, para así poder ver a través de ellas.

Es así que con todos estos módulos se implementó el programa *aquarium_view.py* siguiendo el patrón de diseño Modelo-Vista-Controlador. El controlador maneja la mecánica de movimiento de la cámara, en la parte de modelo se definen algunas constantes, se crean las *gpuShapes* del acuario, los voxeles y los peces, y se definen características como su posición en la escena y el movimiento de su cola. Por último en la vista se procede a dibujar toda la escena.

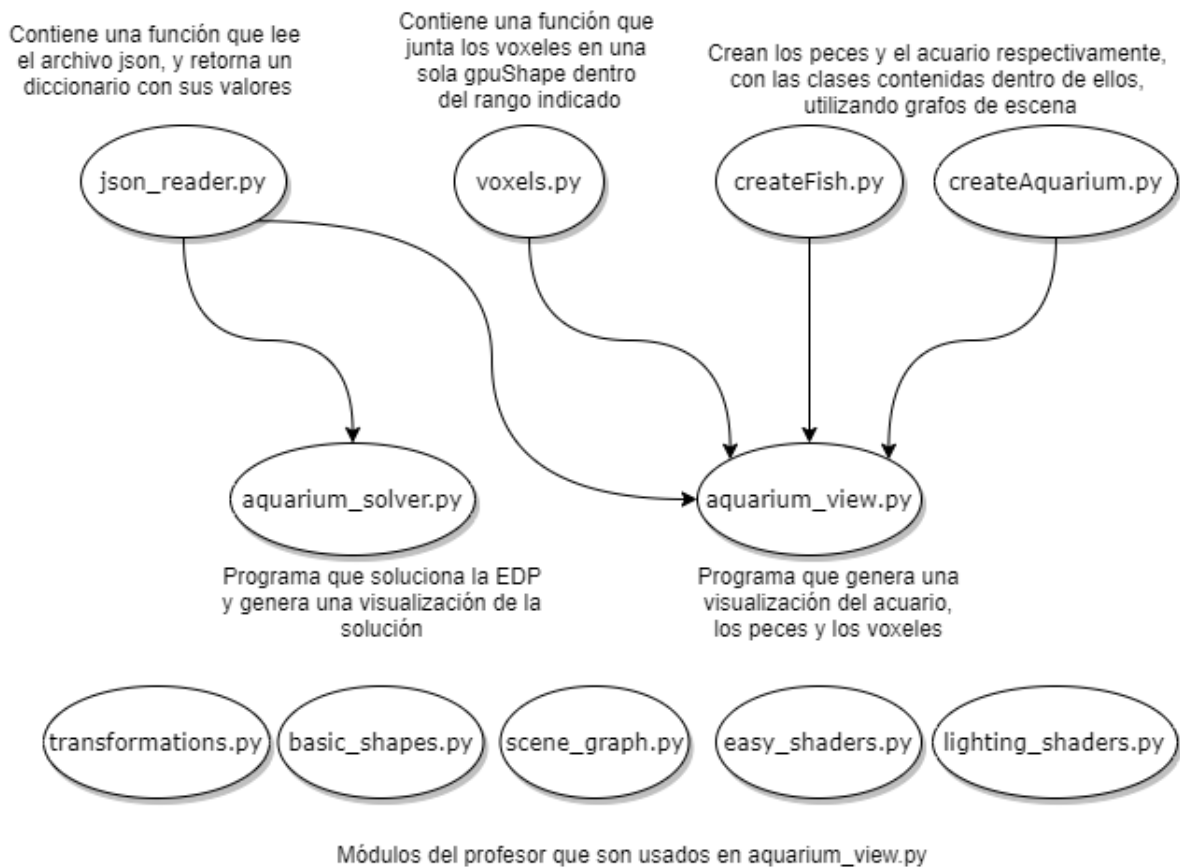


Figura 1: Diagrama con los programas y módulos utilizados.

2. Instrucciones de ejecución y Resultados

Para iniciar el programa *aquarium_solver.py* se debe ingresar un archivo *json* que contenga las diferentes constantes y condiciones del problema. Una vez ejecutado el programa se resolverá la EDP automáticamente y se guardará la solución en un archivo *numpy* con el nombre especificado. Además es posible visualizar la solución de la EDP, ya que se genera un plot con la solución.

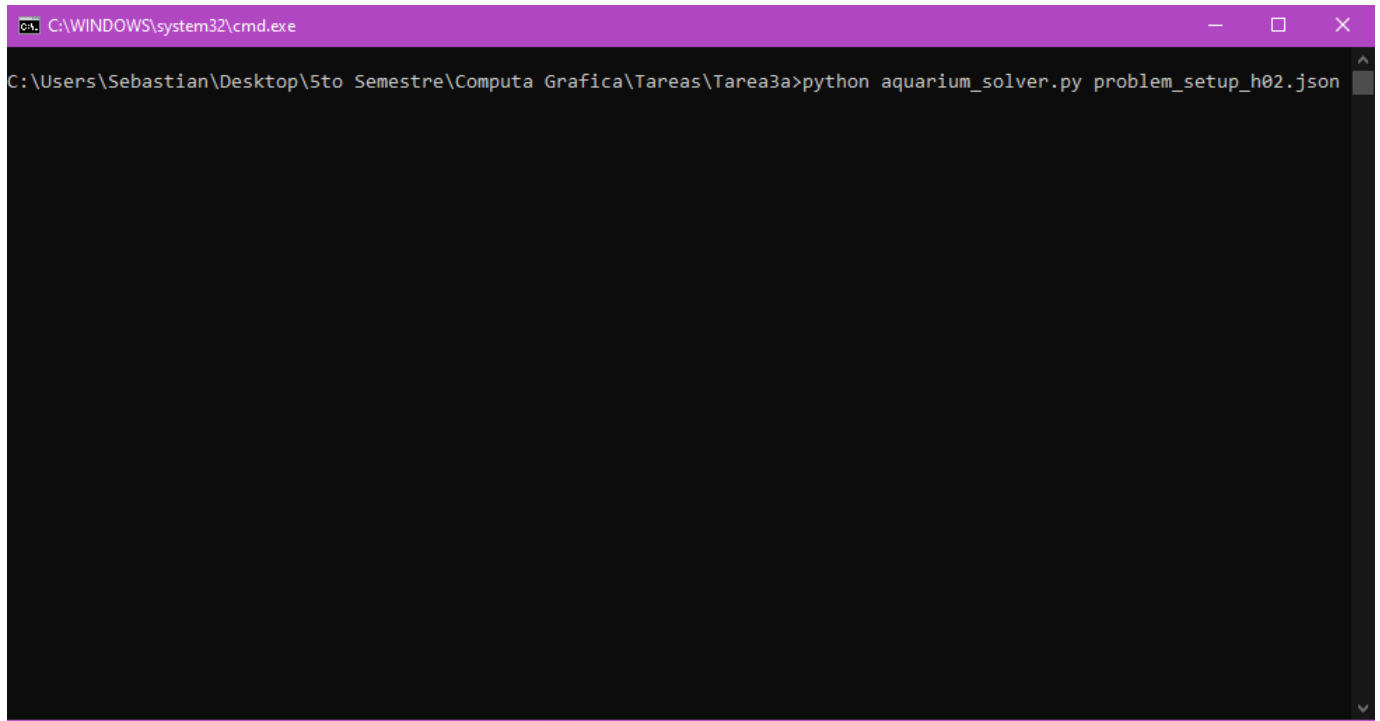


Figura 2: Ejecución *aquarium_solver.py*

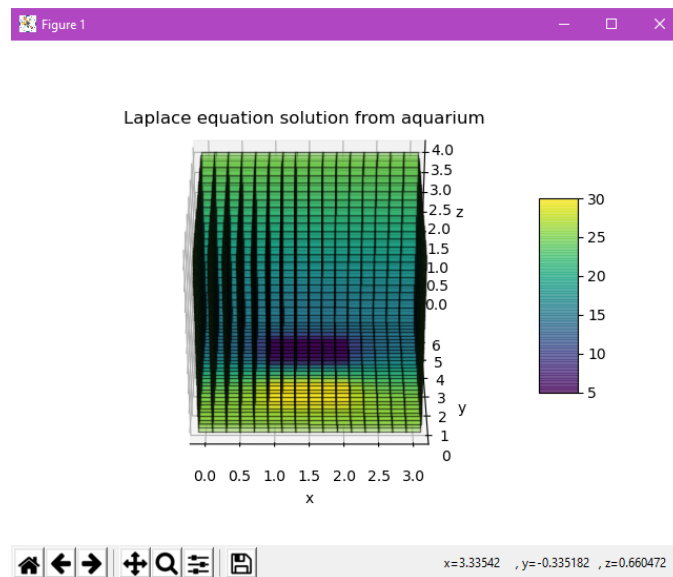
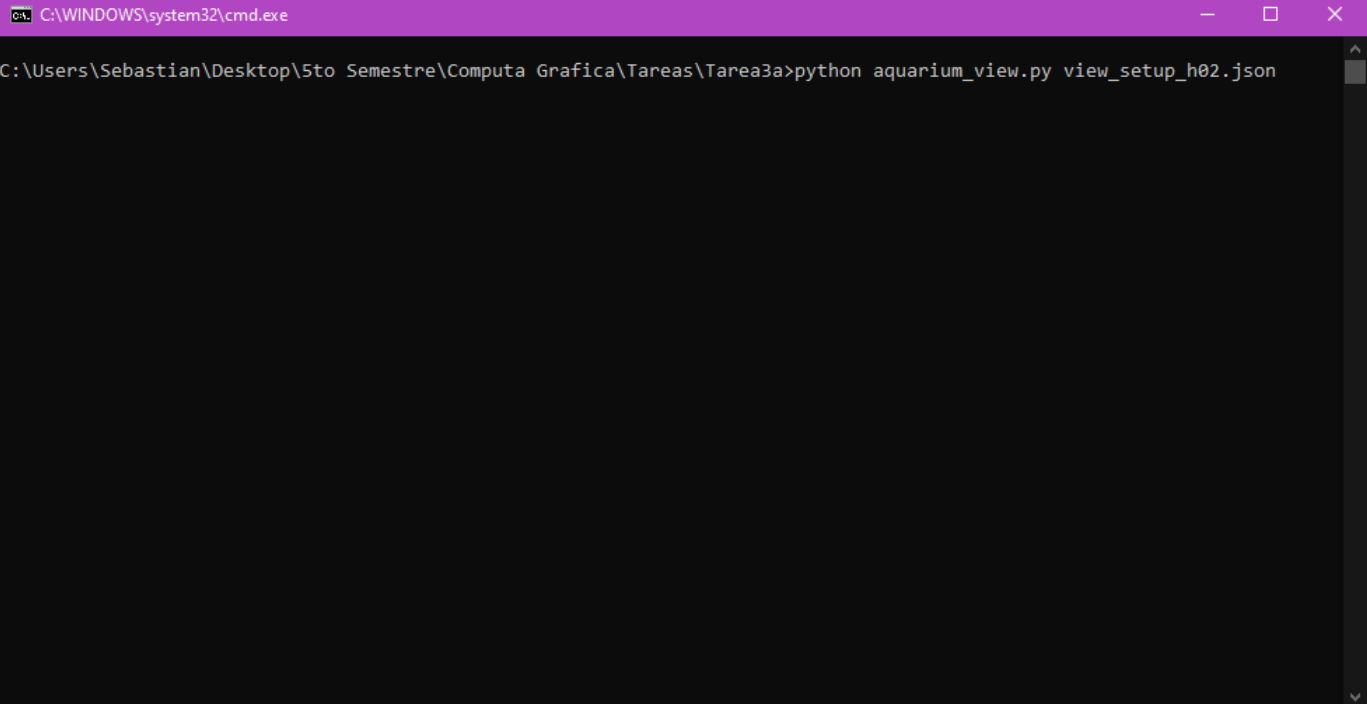


Figura 3: Ejemplo de visualización de la solución de la EDP

Por otro lado, para iniciar el programa *aquarium_view.py* también se debe ingresar un archivo *json*, pero que en este caso contenga las temperaturas ideales, el número de peces y el nombre del archivo de donde se sacarán los voxels para la visualización. El programa comenzará con el acuario en el medio de la escena y dependiendo del archivo ingresado se generarán los peces y los voxels señalando las zonas correspondientes. Cabe mencionar que es posible rotar la cámara presionando las flechas izquierda o derecha, y hacer un zoom-in o zoom-out con las flechas de arriba o abajo. Para visualizar los voxels de las zonas preferidas por cada tipo de pez se debe presionar las letra A, B y C, además si se presiona la D se puede visualizar todos los voxels juntos.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Sebastian\Desktop\5to Semestre\Computa Grafica\Tareas\Tarea3a>python aquarium_view.py view_setup_h02.json
```

Figura 4: Ejecución *aquarium_view.py*

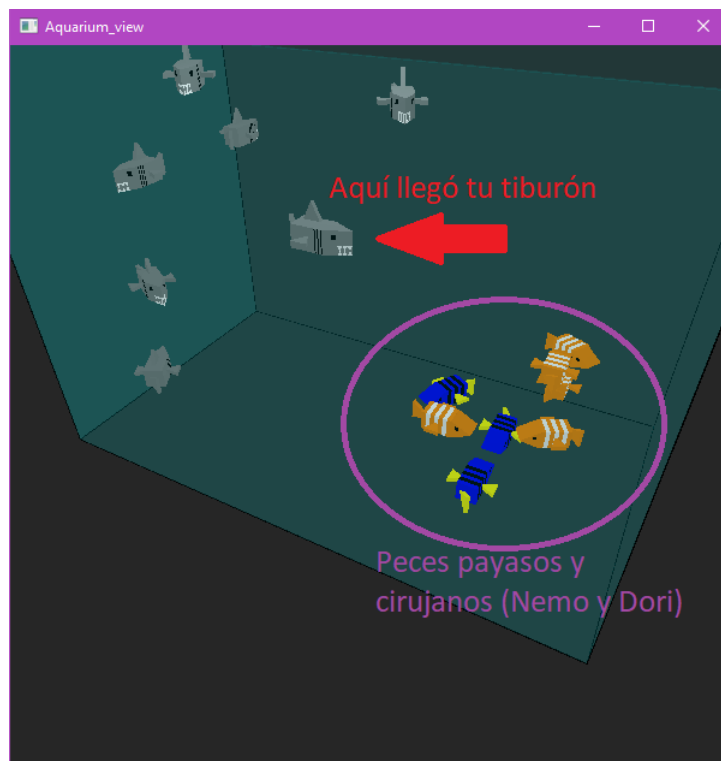


Figura 5: Escena 1 aquarium_view.py

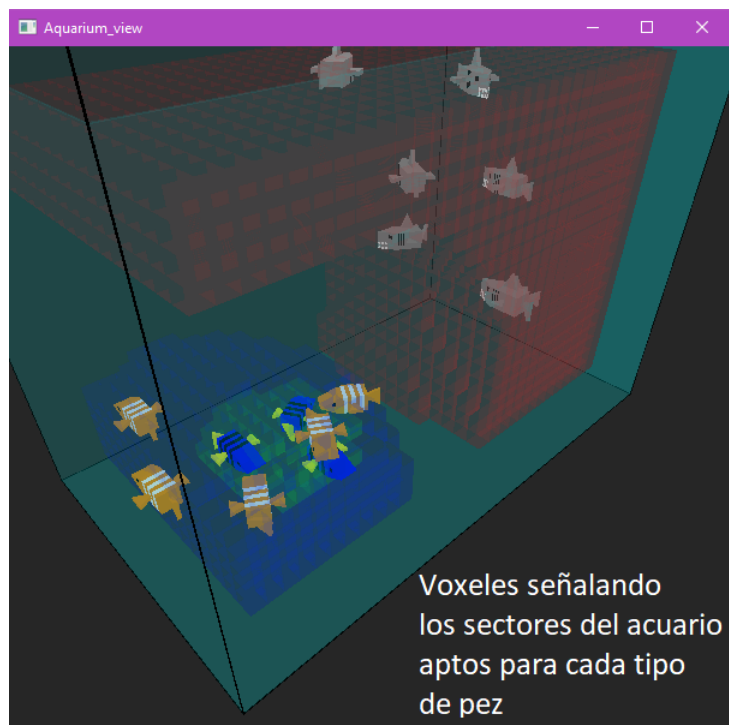


Figura 6: Escena 2 aquarium_view.py