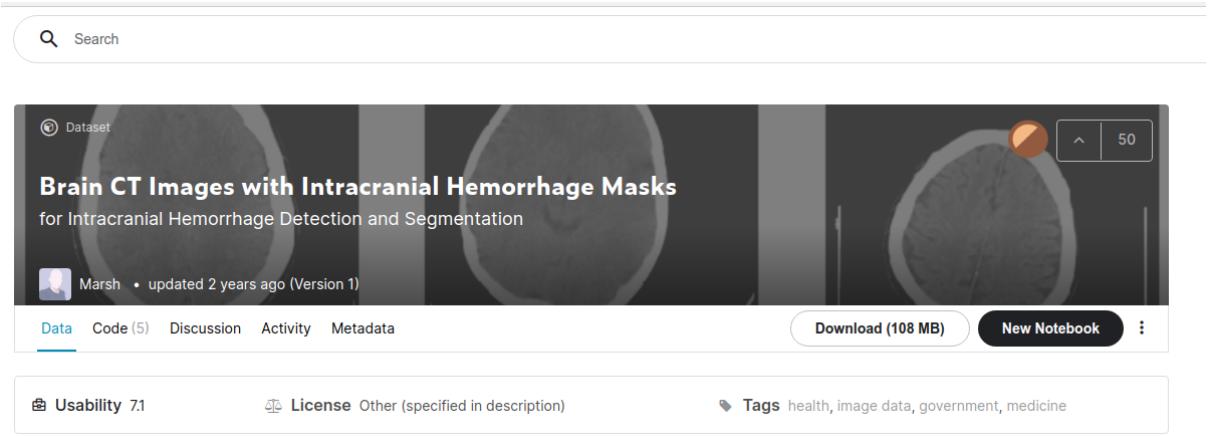
Alumno: Cesar Madera Garces

## Dataset de tomografias:
## https://www.kaggle.com/vbookshelf/computed-tomography-ct-images




## Algoritmo para la obtencion de puntos en el contorno del cerebro: (main.py)

- Abrir las imagenes
- pixeles = []
- Por cada imagen:
    - Ignorar las primeras filas en negro
    - Por cada 3 filas de pixeles:
        - si ya se encontro pixeles “blancos” anteriormente pero en esta iteracion no, terminar el algoritmo, sino:
        - Recorrer de izquierda a derecha y encontrar el primer punto en “blanco”
        - Recorrer de derecha a izquierda y encontrar el primer punto en “blanco”
        - Insertar ambos pixeles a “pixeles” (primero izquierda luego derecha)

*los pixeles blancos se han definido con un threshold entre 10 y 40

```
def binarize(image_to_transform, threshold,iteration):
    output_image=image_to_transform
    modified_image=image_to_transform.convert("L")
    all_points = []
    first_white_pixel =False
    for y in range(0,output_image.height,3):
        points = []
        white_pixel = False
        x1 = 0
        x2= 640
        for x in range(floor(output_image.width/8.0),floor(output_image.width*7/8.0)):
            if modified_image.getpixel((x,y))> threshold and modified_image.getpixel((x,y))< 40: #note that the first parameter is actually a tuple object
                #print(modified_image.getpixel((x,y)))
                output_image.putpixel( (x,y), (255,0,0) )
                first_white_pixel = white_pixel = True
                points.append([x,y,iteration])
                x1 = x
                break
        for x in reversed(range(floor(output_image.width/8.0),floor(output_image.width*7/8.0))):
            if modified_image.getpixel((x,y))> threshold and modified_image.getpixel((x,y))< 40: #note that the first parameter is actually a tuple object
                #print(modified_image.getpixel((x,y)))
                output_image.putpixel( (x,y), (255,0,0) )
                first_white_pixel = white_pixel = True
                points.append([x,y,iteration])
                x2 = x
                break
        if x2-x1 > 550 and first_white_pixel:
            break
        if white_pixel == False and first_white_pixel == True :
            break
        if white_pixel:
            all_points.append(points)
    return all_points,output_image
```

## Algoritmo para la obtencion de los indices para la creacion de la malla de triangulos: (main.py)

- indices= []
- Por cada uno de las listas de pixeles de las imagenes excepto el ultimo:
    - Se obtiene el pixel derecho de la fila actual y la siguiente de la lista actual y la siguiente (en otras palabras los pixeles j y j+1 de la imagen i y i+1 )
    - Se crean un cuadrado en base a dos triangulos con los cuatro puntos anteriormente mencionados (solo se obtienen los indices y se insertan como tripletas en “indices”).
    - Si en caso sobran puntos en cualquiera de los dos ultimos pasos, se repiten los anteriores pasos pero tomando en cuenta los siguientes puntos de la lista faltante y los ultimos puntos de la lista terminada
- Se guardan los resultados de pixeles y indices en un csv para su futuro proceso en el codigo en c++

*se realiza el mismo proceso para pixeles derechos y izquierdos

```
def process_squares(all_images_points,edges,counter_indexes1,counter_indexes2,iteration):
    vertex_array1 = all_images_points[iteration]
    vertex_array2 = all_images_points[iteration+1]
    inferior_limit = min(len(vertex_array1)-1, len(vertex_array2)-1)
    superior_limit = max(len(vertex_array1)-1, len(vertex_array2)-1)
    for j in range(inferior_limit):
        edges.append([counter_indexes1+2*j,counter_indexes1+2*(j+1), counter_indexes2+2*(j)])
        edges.append([counter_indexes1+2*(j+1), counter_indexes2+2*(j),counter_indexes2+2*(j+1)])
    if(inferior_limit == len(vertex_array1)-1):
        for j in range (inferior_limit,superior_limit):
            edges.append([counter_indexes1+2*(inferior_limit-2),counter_indexes1+2*(inferior_limit-1), counter_indexes2+2*(j)])
            edges.append([counter_indexes1+2*(inferior_limit-1), counter_indexes2+2*(j),counter_indexes2+2*(j+1)])
    else:
        for j in range (inferior_limit,superior_limit):
            edges.append([counter_indexes1+2*j,counter_indexes1+2*(j+1), counter_indexes2+2*(inferior_limit-2)])
            edges.append([counter_indexes1+2*(j+1), counter_indexes2+2*(inferior_limit-2),counter_indexes2+2*(inferior_limit-1)])
```

## Algoritmo para la reconstruccion del cerebro: (main.cpp)

- Se leen ambos archivos y se obtienen de vuelta el vector de indices y vertices. ( en este proceso cabe recalcar que se modificaron las posiciones de los vertices para que tenga una mayor concordancia con imagenes reales. En ese sentido, los valores x, y se les multiplico por 0.1 ya que iban desde 80 hasta 560 y solo se contaba con 32 imagenes, por lo que el valor de z iba desde 0 hasta 32)
- Se cargan el vertex shader y fragment shader
- Se pasan los vectores de informacion
- Se crean la matriz de proyeccion, vista y modelo y se procede a dibujar los triangulos.

*imagen anexada al final

## codigos para compilar el proyecto(en terminal)

python3 main.py

g++ main.cpp glew.c stb_image.cpp -lglfw -ldl -lglut -lGL -lGLU -lglfw3
.
./a.out

```
class Fondo
{
public:
  Shader *shader;
  unsigned int texture;
  unsigned int VBO, VAO, EBO;
  vector<float> vertices;
  vector<int> indices;
  void load_file(){
    string filename = "./vertex.txt";
    string filename2 = "./edges.txt";
    ifstream file,file2;
    file.open(filename.c_str(), ios::in);
    file2.open(filename2.c_str(), ios::in);
    float lectura;
    int cont = 0;
    while(!file.eof()){
      file >> lectura;
      if(cont!=2){
        lectura *=  32.0f / 320.0f;
        cont += 1;
      }else{
        cont=0;
      }

      vertices.push_back(lectura);
    }
    file.close();

    while (!file2.eof())
    {
      file2 >> lectura;
      indices.push_back(lectura);
    }
    file2.close();
  }
```

**Imagenes obtenidas:**