Netobook Discretizacao Pulmonar

Versão 1.0

Disciplina: Processo de Negócios e Fundamentos de Sistema da Informação

Autor: Wallace Santos Ribeiro

RA: 309767

Sumário

Prefácio

Este documento possui a finalidade de apresentar ao leitor a etapa de inicialização do software “Netobook Discretizacao Pulmonar” para visualização pulmonar, exibição de suas funcionalidades e interface para uso.

Essa é a versão 1.0 do documento, revisada para garantir uma boa leitura e fácil compreensão do escopo geral do software.

1. Introdução

Em centros médicos, torna-se notório a procura de exames para Raio-X e Tomografia, a fim de diagnosticar possíveis lesões ou fraturas. E a necessidade de diagnósticos desses exames torna-se emergente dependendo da situação. Este software surge como facilitador através da inteligência artificial, tornando-se capaz de verificar alguma inconformidade e apresentar um diagnostico com base na imagem, tornando o processo mais eficaz e eficiente para demanda apresentada.

1.1. Tema

Software de visualização pulmonar para diagnostico do paciente.

1.2. Objetivo do Projeto

Apresentar um software que realiza a renderização da tomografia a partir do esqueleto e tecido interno do paciente.

2. Descrição Software

2.1. Requisitos Funcionais

Código	Descrição
RFU.001	Acessar ficha do paciente.
RFU.002	Executar renderização da Tomografia.
RFU,003	Alterar perspectiva de visualização da Tomografia (Pulmões, tecidos moles, ou ossos).

2.2. Requisitos não Funcionais

Código	Descrição
RNFU.001	O **load_scan** realiza a busca da imagem DICOM no diretório e carrega as fatias da tomografia.
RNFU.002	O pydicom.dcmread() lê cada imagem.
RNFU.003	Com o **input_folder** é feito a busca na lista dos pacientes.
RNFU.004	Realizada a conversão de imagem DICOM para formato HU através da função **def get_pixels_hu(scans).**
RNFU.005	Após a função **dep plot_all_slices**, recebe o volume 3D da tomografia (**image_data**) e plota todos os cortes (fatias) em uma grade única.
RNFU.006	Se o threshold for **-300**, será visualizado apenas o **pulmão** (pois HU ≈ -700). Se for **700**, verá apenas **ossos**.
RNFU.007	A renderização será carregada após a regularização do eixo e escala do modelo 3D.

2.3. Caso de Uso

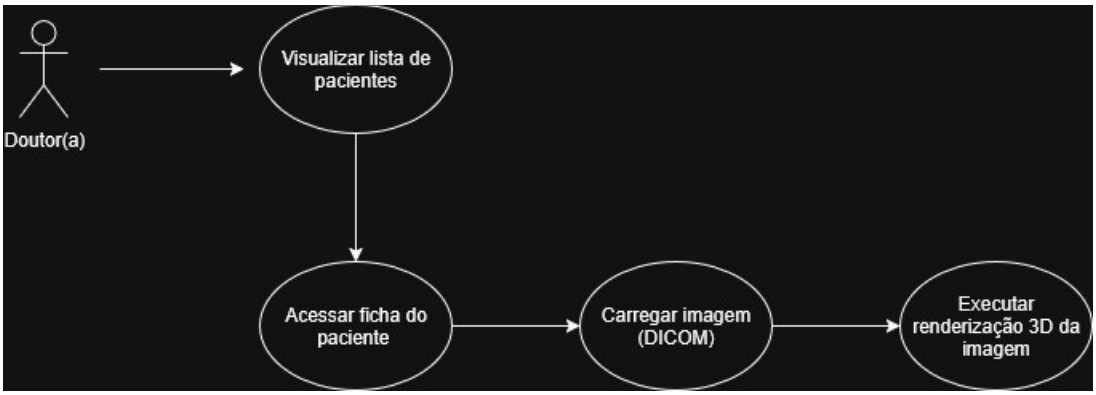

Diagrama caso de uso: Doutor(a) executando renderização da tomografia.

3. Processo de Software

3.1. Código


```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "cd091a7a-d885-4623-8209-70b380dd9a4d",
   "metadata": {},
   "outputs": [],
   "source": [
    "pip install pydicom"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "0aadaf4b-0df7-4b80-adff-4aec642a0a84",
   "metadata": {},
   "outputs": [],
   "source": [
    "%matplotlib inline\n",
    "\n",
    "import numpy as np # linear algebra\n",
    "import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)\n",
    "import pydicom\n",
    "import os\n",
    "import scipy.ndimage\n",
    "import matplotlib.pyplot as plt\n",
    "\n",
    "from skimage import measure, morphology\n",
    "from mpl_toolkits.mplot3d.art3d import Poly3DCollection\n",
    "\n",
    "# Some constants \n",
    "INPUT_FOLDER = './CTScans/manifest-1557326747206/LCTSC/'\n",
    "patients = os.listdir(INPUT_FOLDER)\n",
    "patients.sort()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
```

```
  "id": "68595e3e-58ba-4795-8140-cb717645d89e",
  "metadata": {},
  "outputs": [],
  "source": [
   "def load_scan(path):\n",
   "    slices = []\n",
   "    for s in os.listdir(path):\n",
   "        try:\n",
   "            slice = pydicom.dcmread(os.path.join(path, s))\n",
   "            slices.append(slice)\n",
   "        except Exception as e:\n",
   "            print(f\"Erro ao carregar {s}: {e}\")\n",
   "    slices.sort(key=lambda x: int(x.InstanceNumber))\n",
   "    # Calcular slice_thickness\n",
   "    if len(slices) > 1:\n",
   "        try:\n",
   "            slice_thickness = abs(slices[0].ImagePositionPatient[2] -
slices[1].ImagePositionPatient[2])\n",
   "        except:\n",
   "            slice_thickness = abs(slices[0].SliceLocation - slices[1].SliceLocation)\n",
   "        for s in slices:\n",
   "            s.SliceThickness = slice_thickness\n",
   "    return slices"
  ]
 },
```

Código tópico 1.

O início do código, possui a célula "**cell**", que executa a instalação automática do "pip install pydicom". Após essa etapa, as seguintes bibliotecas são acionadas, sendo: **numpy e bandas** (manipulação de dados), **pydicom** (leitura de imagens DICOM), **scipy.ndimage** (processamento de imagens), **matplotlib / skimage / mpl_toolkits.mplot3d** (visualização 3D). Sendo o INPUT_FOLDER, responsável por procurar os arquivos em formato DICOM e o patients = os.listdir(INPUT_FOLDER) em exibir a lista dos pacientes. O, patients.sort() organiza a lista dos pacientes, facilitando a identificação. Por fim, o pydicom.dcmread() é responsável por ler cada imagem DICOM, onde é adicionado o Array “slices” para retorno, e calculo da **espessura entre fatias** (slice_thickness), necessária para reconstruir o modelo 3D corretamente.

```
{
  "cell_type": "code",
  "execution_count": 5,
  "id": "ca1a5fb4-3b5d-4d46-a801-2022e49ce213",
  "metadata": {},
  "outputs": [],
  "source": [
   "def get_pixels_hu(scans):\n",
   "    image = np.stack([s.pixel_array for s in scans])\n",
   "    # Convert to int16 (from sometimes int16), \n",
   "    # should be possible as values should always be low enough (<32k)\n",
   "    image = image.astype(np.int16)\n",
   "\n",
   "    # Set outside-of-scan pixels to 0\n",
   "    # The intercept is usually -1024, so air is approximately 0\n",
   "    image[image == -2000] = 0\n",
   "    \n",
   "    # Convert to Hounsfield units (HU)\n",
   "    intercept = scans[0].RescaleIntercept\n",
   "    slope = scans[0].RescaleSlope\n",
   "    \n",
   "    if slope != 1:\n",
   "        image = slope * image.astype(np.float64)\n",
   "        image = image.astype(np.int16)\n",
   "        \n",
   "    image += np.int16(intercept)\n",
   "    \n",
   "    return np.array(image, dtype=np.int16)"
  ]
 },
```

Código Tópico 2

Nesta parte do código, é realizado a corversão de dados brutos do **DICOM** (arquivos da tomografia) para **Unidades de Hounsfield (HU)**, que representam a densidade dos tecidos no corpo humano. Para realizar essa ação, a função “**def get_pixels_hu(scans)**” percorre aas fátias DICOM (scans), o “pixel_array” extrai a imagem 2D da fatia, onde é feita a junção dentro de “image.shape”.

- **image = image.astype(np.int16)** - Garante que todos os pixels estejam no tipo int16.
- **return np.array(image, dtype=np.int16)** – Retorna a matriz 3D com as intensidades de Hounsfield (HU).

```
 "source": [
   "import os\n",
   "import numpy as np\n",
   "import pydicom\n",
   "\n",
   "def load_scan(path):\n",
   "    slices = []\n",
   "    if not os.listdir(path):  # Checa se a pasta está vazia\n",
   "        print(\"A pasta não contém arquivos.\")\n",
   "        return []\n",
   "        \n",
   "    for s in os.listdir(path):\n",
   "        try:\n",
   "            slice = pydicom.dcmread(os.path.join(path, s))\n",
   "            slices.append(slice)\n",
   "        except Exception as e:\n",
   "            print(f\"Erro ao carregar {s}: {e}\")\n",
   "    \n",
   "    if not slices:  # Se nenhum arquivo DICOM válido foi carregado\n",
   "        print(\"Nenhum arquivo DICOM válido foi carregado.\")\n",
   "        return []\n",
   "\n",
   "    slices.sort(key=lambda x: int(x.InstanceNumber))\n",
   "\n",
   "    # Calcula a espessura do slice se houver pelo menos dois arquivos\n",
   "    if len(slices) > 1:\n",
   "        try:\n",
   "            slice_thickness = abs(slices[0].ImagePositionPatient[2] -
slices[1].ImagePositionPatient[2])\n",
   "        except:\n",
   "            slice_thickness = abs(slices[0].SliceLocation - slices[1].SliceLocation)\n",
   "        for s in slices:\n",
   "            s.SliceThickness = slice_thickness\n",
   "            \n",
   "    return slices\n",
   "\n",
   "# Após isso, execute novamente o trecho principal:\n",
   "try:\n",
   "    first_patient = load_scan(INPUT_FOLDER + patients[0])\n",
   "    if first_patient:  # Verifica se a lista não está vazia antes de prosseguir\n",
   "        first_patient_pixels = get_pixels_hu(first_patient)\n",
   "        \n",
   "        # Código para exibir o histograma e as imagens\n",
   "        plt.hist(first_patient_pixels.flatten(), bins=80, color='c')\n",
   "        plt.xlabel(\"Hounsfield Units (HU)\")\n",
   "        plt.ylabel(\"Frequency\")\n",
   "        plt.show()\n",
```

```
  "        \n",
  "        # Exibe uma fatia\n",
  "        plt.imshow(first_patient_pixels[80], cmap=plt.cm.gray)\n",
  "        plt.show()\n",
  "    else:\n",
  "        print(\"Não foi possível carregar os dados do paciente.\")\n",
  "except Exception as e:\n",
  "    print(f\"Erro durante o processamento: {e}\")\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": 11,
 "id": "67b797af-74b3-435b-9490-cd8112102233",
 "metadata": {},
 "outputs": [],
 "source": [
  "def resample(image, scan, new_spacing=[1,1,1]):\n",
  "    # Determina o espaçamento atual de pixels\n",
  "    try:\n",
  "        slice_thickness = scan[0].SliceThickness\n",
  "        pixel_spacing = scan[0].PixelSpacing\n",
  "        # Garantindo que PixelSpacing seja uma lista de valores flutuantes\n",
  "        spacing = np.array([float(slice_thickness)] + list(map(float, pixel_spacing)))\n",
  "    except AttributeError:\n",
  "        print(\"Erro: SliceThickness ou PixelSpacing ausente no arquivo DICOM.\")\n",
  "        return None, None\n",
  "\n",
  "    # Calcula o fator de redimensionamento\n",
  "    resize_factor = spacing / new_spacing\n",
  "    new_real_shape = image.shape * resize_factor\n",
  "    new_shape = np.round(new_real_shape)\n",
  "    real_resize_factor = new_shape / image.shape\n",
  "    new_spacing = spacing / real_resize_factor\n",
  "    \n",
  "    # Redimensiona a imagem\n",
  "    image = scipy.ndimage.zoom(image, real_resize_factor, mode='nearest')\n",
  "    \n",
  "    return image, new_spacing\n"
 ]
},
```

Código Tópico 3

Agora será realizado o carregamento das fatias da tomografia com “load_scene” com base na pasta do paciente, para leitura, será utilizado “pydicom.dcmread()”. Se não tiver

imagem, retornará como vazio. Para garantir a estrutura correta do modelo 3D, a função slices.sort é utilizada.

- **if len(slices) > 1:** - Calcula a espessura entre fatias (SliceThickness), necessária para reconstruir o volume 3D corretamente em escala real.
- **first_patient =** load_scan(INPUT_FOLDER + patients[0]) - carrega o primeiro paciente da lista e converte as fatias para Unidades de Hounsfield (HU).
- **plt.xlabel("Hounsfield Units (HU)") -** Cria um histograma dos valores HU**.**
- **plt.imshow(first_patient_pixels[80], cmap=plt.cm.gray) –** Exibe uma fatia 2D da tomografia.

- **image = scipy.ndimage.zoom(image, real_resize_factor, mode='nearest') -** Usa o scipy.ndimage.zoom() para redimensionar o volume 3D de acordo com o fator calculado.
- **return image, new_spacing -** O novo espaçamento (confirmado após arredondamento).

```
{
 "cell_type": "code",
 "execution_count": 13,
 "id": "dc748da2-63cd-413e-8f87-e86db7729cfd",
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Shape before resampling\t (130, 512, 512)\n",
    "Shape after resampling\t (390, 500, 500)\n"
   ]
  }
 ],
 "source": [
  "pix_resampled, spacing = resample(first_patient_pixels, first_patient, [1,1,1])\n",
  "print(\"Shape before resampling\\t\", first_patient_pixels.shape)\n",
  "print(\"Shape after resampling\\t\", pix_resampled.shape)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 15,
 "id": "95ed6c04-3cc4-428f-9470-edc47d6e9a9a",
 "metadata": {},
 "outputs": [],
 "source": [
  "from skimage import measure\n",
  "from mpl_toolkits.mplot3d.art3d import Poly3DCollection\n",
  "import matplotlib.pyplot as plt\n",
  "\n",
  "def plot_3d(image, threshold=0):\n",
  "    # Posiciona a imagem de forma vertical\n",
  "    p = image.transpose(2, 1, 0)\n",
  "\n",
  "    # Adapta para capturar todos os valores retornados pela função\n",
  "    verts, faces, _, _ = measure.marching_cubes(p, threshold)\n",
  "\n",
  "    fig = plt.figure(figsize=(10, 10))\n",
  "    ax = fig.add_subplot(111, projection='3d')\n",
  "\n",
  "    # Indexação para gerar uma coleção de triângulos\n",
  "    mesh = Poly3DCollection(verts[faces], alpha=0.70)\n",
  "    face_color = [0.45, 0.45, 0.75]\n",
  "    mesh.set_facecolor(face_color)\n",
  "    ax.add_collection3d(mesh)\n",
```

```
  "\n",
  "    ax.set_xlim(0, p.shape[0])\n",
  "    ax.set_ylim(0, p.shape[1])\n",
  "    ax.set_zlim(0, p.shape[2])\n",
  "\n",
  "    plt.show()\n"
 ]
},
```

Código Tópico 4

Esse é a etapa final para renderização 3D da Tomografia, reconstituída a partir das fatias DICOM.

- **pix_resampled, spacing = resample(first_patient_pixels, first_patient, [1,1,1]) -** Ajusta o espaçamento entre pixels e fatias da tomografia.
- **skimage.measure** → possui o algoritmo marching cubes (reconstrói superfícies 3D).
- **Poly3DCollection** → cria a malha 3D.
- **matplotlib** → exibe o resultado em uma figura interativa.
- **measure.marching_cubes() -** percorre o volume voxel por voxel e cria uma malha 3D (mesh).
- **plt.show() -** Exibe o modelo 3D renderizado.

```
{
 "cell_type": "code",
 "execution_count": null,
 "id": "d0d119b8-706c-4b6f-bf3f-d3668d389da8",
 "metadata": {
  "tags": []
 },
 "outputs": [],
 "source": [
  "MIN_BOUND = -1000.0\n",
  "MAX_BOUND = 400.0\n",
  "    \n",
  "def normalize(image):\n",
  "    image = (image - MIN_BOUND) / (MAX_BOUND - MIN_BOUND)\n",
  "    image[image>1] = 1.\n",
  "    image[image<0] = 0.\n",
  "    return image"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "c6105339-e60b-42e1-8693-eb9ba6726a80",
 "metadata": {},
 "outputs": [],
 "source": [
  "PIXEL_MEAN = 0.25\n",
  "\n",
  "def zero_center(image):\n",
  "    image = image - PIXEL_MEAN\n",
  "    return image"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "79c6d920-6e17-493d-9812-d35f3fc9d4eb",
 "metadata": {
  "scrolled": true
 },
 "outputs": [],
 "source": [
  "import matplotlib.pyplot as plt\n",
  "import math\n",
  "\n",
  "def plot_all_slices(image_data, slices_per_row=8):\n",
  "    \"\"\"\n",
```

```
  "    Plota todos os cortes de uma tomografia em uma grade única.\n",
  "    \n",
  "    Parameters:\n",
  "    - image_data: numpy array contendo a imagem de tomografia (3D).\n",
  "    - slices_per_row: número de cortes por linha na grade.\n",
  "    \"\"\"\n",
  "    num_slices = image_data.shape[0]\n",
  "    num_rows = math.ceil(num_slices / slices_per_row)\n",
  "    \n",
  "    fig, axes = plt.subplots(num_rows, slices_per_row, figsize=(15, num_rows * 2))\n",
  "    axes = axes.flatten()  # Transformar a matriz de eixos em uma lista\n",
  "    \n",
  "    for i in range(num_slices):\n",
  "        axes[i].imshow(image_data[i], cmap=\"gray\")\n",
  "        axes[i].axis(\"off\")\n",
  "        axes[i].set_title(f\"Corte {i+1}\")\n",
  "\n",
  "    # Desliga os eixos que não têm imagem para mostrar\n",
  "    for i in range(num_slices, len(axes)):\n",
  "        axes[i].axis(\"off\")\n",
  "    \n",
  "    plt.tight_layout()\n",
  "    plt.show()\n",
  "\n",
  "# Use a função com seus dados\n",
  "plot_all_slices(first_patient_pixels)\n"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "d7b589bb-38d1-4d5e-969a-e5eab2baa1b5",
 "metadata": {},
 "outputs": [],
 "source": [
  "from skimage import measure\n",
  "from mpl_toolkits.mplot3d.art3d import Poly3DCollection\n",
  "import matplotlib.pyplot as plt\n",
  "import numpy as np\n",
  "\n",
  "def plot_3d_reconstruction(image_data, threshold=-300):\n",
  "    \"\"\"\n",
  "    Reconstrói e plota o modelo 3D da tomografia com tons de cinza.\n",
  "    \n",
  "    Parameters:\n",
  "    - image_data: numpy array 3D contendo os cortes da tomografia.\n",
  "    - threshold: valor limite para segmentar o volume (define o contorno do corpo).\n",
```

```
   "    \"\"\"\n",
   "    # Transposição para alinhar o volume corretamente\n",
   "    p = image_data.transpose(2, 1, 0)\n",
   "\n",
   "    # Extração dos vértices e faces para o modelo 3D\n",
   "    verts, faces, _, values = measure.marching_cubes(p, threshold)\n",
   "    \n",
   "    # Mapear os valores de intensidade para tons de cinza\n",
   "    face_colors = plt.cm.gray((values - values.min()) / (values.max() - values.min()))\n",
   "    \n",
   "    # Plotando a superfície 3D com tons de cinza\n",
   "    fig = plt.figure(figsize=(10, 10))\n",
   "    ax = fig.add_subplot(111, projection='3d')\n",
   "    \n",
   "    # Configuração do modelo 3D com cores em tons de cinza\n",
   "    mesh = Poly3DCollection(verts[faces], facecolors=face_colors, alpha=0.7)\n",
   "    ax.add_collection3d(mesh)\n",
   "\n",
   "    # Ajuste dos limites para o volume do modelo\n",
   "    ax.set_xlim(0, p.shape[0])\n",
   "    ax.set_ylim(0, p.shape[1])\n",
   "    ax.set_zlim(0, p.shape[2])\n",
   "\n",
   "    plt.show()\n",
   "\n",
   "# Chame a função com os dados da tomografia\n",
   "plot_3d_reconstruction(first_patient_pixels)\n"
  ]
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Python 3 (ipykernel)",
  "language": "python",
  "name": "python3"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.12.4"
```

```
 }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Neste segmento de código, após a renderização, representa a fase de normalização, centralização e visualização 2D/3D das imagens de tomografia computadorizada (TC) após serem convertidas em Hounsfield Units (HU).

- **def normalize(image): -** Converte os valores de Hounsfield Units (HU).
- **PIXEL_MEAN = 0.25 -** Subtrai a média global (0.25) de todos os pixels, de modo que as intensidades fiquem centradas em torno de zero.
- **def plot_all_slices(image_data, slices_per_row=8): -** Recebe o volume 3D da tomografia (image_data) e plota todos os cortes (fatias) em uma grade única.
- **def plot_3d_reconstruction(image_data, threshold=-300): -** Cria uma reconstrução 3D da tomografia a partir das fatias 2D atráves do algoritmo Marching Cubes do scikit-image para extrair vértices (verts) e faces (faces) que formam uma superfície 3D.

4. Glossário

Página	Palavra	Descrição
6.	Conversão	Alteração da propriedade base de algo.
6.	DICOM	Formato de imagem de Tomografia.
7.	Diagrama	Esquema que representa um processo de maneira visual.
21.	Faces	Preenchimento presente em todos os lados de um objeto.
15.	Histograma	Conjunto de retângulos que têm as bases sobre o eixo *x.*
21.	Marching Cubes	Método de computação gráfica que extrai uma malha poligonal de superfície 3D de um campo escalar volumétrico.
15.	Redimensionar	Alteraração de escala e proporção de algo.
5.	Renderização	Emulação de uma cena 3D em formato de imagem.
2..	Software	Sistema executável no computador.
17.	Voxel	Unidade cúbica que representa a menor porção de um volume tridimensional.
21.	Vértices	É o ponto exato onde duas ou mais linhas, arestas ou curvas se encontram, formando um ângulo ou uma "quina"