you are writing a desktop GUI in python thai will give you 5 codes for 3d visualization for segmentations each code from an ai model and i need to collect them to represnt them in GUI and you must make organ and model each of them choosable from dropdown (organs : lungs , kidney, liver) (ai models : MedSam,TotalSegmentator,UNest)

#lungs : (from TotalSegmentator)

```
import os
import nibabel as nib
import numpy as np
from skimage import measure
import pyvista as pv

# ============================
# PATH TO YOUR FOLDER
# ============================
base_dir = r"C:\Users\Asus\Downloads\segmentations (6)"

# ============================
# LUNG PART FILES (5 parts)
# ============================
part_files = {
    "lung_lower_lobe_right.nii.gz": "Right Lower Lobe",
    "lung_middle_lobe_right.nii.gz": "Right Middle Lobe",
    "lung_upper_lobe_right.nii.gz": "Right Upper Lobe",
    "lung_lower_lobe_left.nii.gz": "Left Lower Lobe",
    "lung_upper_lobe_left.nii.gz": "Left Upper Lobe",
}

# ============================
# COLORS
# ============================
colors = {
    "lung_lower_lobe_right.nii.gz": (1.0, 0.1, 0.1),     # Red
    "lung_middle_lobe_right.nii.gz": (0.1, 0.9, 0.1),    # Green
    "lung_upper_lobe_right.nii.gz": (0.1, 0.4, 0.9),     # Blue
    "lung_lower_lobe_left.nii.gz": (1.0, 0.6, 0.1),      # Orange
    "lung_upper_lobe_left.nii.gz": (0.7, 0.2, 0.9),      # Purple
}

# ============================
# CREATE PLOTTER
# ============================
plotter = pv.Plotter()
plotter.add_text("Lung Segmentation — 5 Parts", font_size=14)
plotter.add_axes(line_width=1)
plotter.add_bounding_box(color="black")

actors = {}
```

```
# ============================
# LOAD EACH LOBE & CREATE MESH
# ============================
for fname, label in part_files.items():
    path = os.path.join(base_dir, fname)

    if not os.path.exists(path):
        print(f"⚠ File not found: {fname}")
        continue

    # --- Load NIfTI ---
    nii = nib.load(path)
    mask = nii.get_fdata()
    mask = (mask > 0).astype(np.uint8)

    # --- Generate surface mesh using marching cubes ---
    try:
        verts, faces, _, _ = measure.marching_cubes(mask, level=0.5)
    except ValueError:
        print(f"⚠ Skipping {label} — empty or invalid mask.")
        continue

    # --- Convert to world coordinates (correct orientation) ---
    verts = nib.affines.apply_affine(nii.affine, verts)

    # --- Build mesh ---
    faces = np.hstack([np.full((faces.shape[0], 1), 3), faces]).astype(np.int32)
    mesh = pv.PolyData(verts, faces)

    # --- Add mesh to plotter ---
    actor = plotter.add_mesh(
        mesh,
        color=colors[fname],
        opacity=0.6,
        smooth_shading=True,
        name=label
    )
    actors[fname] = actor

# ============================
# ADD LEGEND
# ============================
legend_entries = [(label, colors[fname]) for fname, label in part_files.items()]
plotter.add_legend(legend_entries, bcolor="white", face="circle", size=(0.25, 0.25))

# ============================
# OPACITY SLIDER
# ============================
def set_opacity(val):
    val = float(val)
    for a in actors.values():
        a.GetProperty().SetOpacity(val)
    plotter.render()

plotter.add_slider_widget(
    callback=set_opacity,
    rng=[0.1, 1.0],
    value=0.6,
    title="Opacity",
    pointa=(0.02, 0.08),
    pointb=(0.35, 0.08)
)

# ============================
# CHECKBOXES TO TOGGLE LOBES
# ============================
y0 = 0.80
dy = 0.07

def make_toggle(fname):
    def _cb(state):
        if fname in actors:
            actors[fname].SetVisibility(bool(state))
            plotter.render()
    return _cb

for i, (fname, label) in enumerate(part_files.items()):
    if fname in actors:
        plotter.add_checkbox_button_widget(
            make_toggle(fname),
            value=True,
            position=(10, int(plotter.window_size[1]*(y0 - i*dy))),
            size=25,
            color_on=colors[fname],
            color_off=(0.7, 0.7, 0.7),
            border_size=1
        )
        plotter.add_text(
            f" {label}",
            position=(40, int(plotter.window_size[1]*(y0 - i*dy))+5),
            font_size=10,
            color="black"
        )

# ============================
# SHOW
# ============================
plotter.show_grid()
plotter.show()
```

#liver : (from TotalSegmentator)

```
import os
import nibabel as nib
import numpy as np
from skimage import measure
import pyvista as pv

# ============================
# PATH TO YOUR FOLDER
# ============================
base_dir = r"C:\Users\Asus\Downloads\total segmentator masks\seg liver ts"

# ============================
# LIVER SEGMENT FILES (8 segments)
# ============================
segment_files = {
    "liver_segment_1.nii.gz": "Segment 1 (Caudate)",
    "liver_segment_2.nii.gz": "Segment 2 (Left Lateral)",
    "liver_segment_3.nii.gz": "Segment 3 (Left Medial)",
    "liver_segment_4.nii.gz": "Segment 4 (Left Medial Superior)",
    "liver_segment_5.nii.gz": "Segment 5 (Right Anterior Inferior)",
    "liver_segment_6.nii.gz": "Segment 6 (Right Posterior Inferior)",
    "liver_segment_7.nii.gz": "Segment 7 (Right Posterior Superior)",
    "liver_segment_8.nii.gz": "Segment 8 (Right Anterior Superior)",
}

# ============================
# COLORS (distinct colors for each segment)
# ============================
colors = {
    "liver_segment_1.nii.gz": (0.9, 0.2, 0.2),     # Red
    "liver_segment_2.nii.gz": (0.2, 0.8, 0.2),     # Green
    "liver_segment_3.nii.gz": (0.2, 0.4, 0.9),     # Blue
    "liver_segment_4.nii.gz": (0.9, 0.9, 0.2),     # Yellow
    "liver_segment_5.nii.gz": (0.9, 0.5, 0.2),     # Orange
    "liver_segment_6.nii.gz": (0.7, 0.2, 0.9),     # Purple
    "liver_segment_7.nii.gz": (0.2, 0.9, 0.9),     # Cyan
    "liver_segment_8.nii.gz": (0.9, 0.2, 0.7),     # Magenta
}

# ============================
# CREATE PLOTTER
# ============================
plotter = pv.Plotter()
plotter.add_text("Liver Segmentation — 8 Segments (Couinaud Classification)",
font_size=14)
plotter.add_axes(line_width=1)
plotter.add_bounding_box(color="black")

actors = {}

# ============================
# LOAD EACH SEGMENT & CREATE MESH
# ============================
for fname, label in segment_files.items():
    path = os.path.join(base_dir, fname)

    if not os.path.exists(path):
        print(f"⚠ File not found: {fname}")
        continue

    # --- Load NIfTI ---
    print(f"Loading {fname}...")
    nii = nib.load(path)
    mask = nii.get_fdata()
    mask = (mask > 0).astype(np.uint8)

    # --- Generate surface mesh using marching cubes ---
    try:
        verts, faces, _, _ = measure.marching_cubes(mask, level=0.5)
        print(f"  {label}: {len(verts)} vertices, {len(faces)} faces")
    except ValueError:
        print(f"⚠ Skipping {label} — empty or invalid mask.")
        continue

    # --- Convert to world coordinates (correct orientation) ---
    verts = nib.affines.apply_affine(nii.affine, verts)

    # --- Build mesh ---
    faces = np.hstack([np.full((faces.shape[0], 1), 3), faces]).astype(np.int32)
    mesh = pv.PolyData(verts, faces)

    # --- Add mesh to plotter ---
    actor = plotter.add_mesh(
        mesh,
        color=colors[fname],
        opacity=0.7,
        smooth_shading=True,
        name=label
    )
    actors[fname] = actor

# ============================
# ADD LEGEND
# ============================
legend_entries = [(label, colors[fname]) for fname, label in segment_files.items() if fname in
actors]
plotter.add_legend(legend_entries, bcolor="white", face="circle", size=(0.3, 0.3))

# ============================
# OPACITY SLIDER
# ============================
def set_opacity(val):
    val = float(val)
    for a in actors.values():
        a.GetProperty().SetOpacity(val)
    plotter.render()

plotter.add_slider_widget(
    callback=set_opacity,
    rng=[0.1, 1.0],
    value=0.7,
    title="Opacity",
    pointa=(0.02, 0.08),
    pointb=(0.35, 0.08)
)

# ============================
# CHECKBOXES TO TOGGLE SEGMENTS
# ============================
y0 = 0.88
dy = 0.055

def make_toggle(fname):
    def _cb(state):
        if fname in actors:
            actors[fname].SetVisibility(bool(state))
            plotter.render()
    return _cb

for i, (fname, label) in enumerate(segment_files.items()):
    if fname in actors:
        plotter.add_checkbox_button_widget(
            make_toggle(fname),
            value=True,
            position=(10, int(plotter.window_size[1]*(y0 - i*dy))),
            size=25,
            color_on=colors[fname],
            color_off=(0.7, 0.7, 0.7),
            border_size=1
        )
        plotter.add_text(
            f" {label}",
            position=(40, int(plotter.window_size[1]*(y0 - i*dy))+5),
            font_size=9,
            color="black"
        )

# ============================
# SHOW ALL / HIDE ALL BUTTONS
# ============================
def show_all(state):
    for a in actors.values():
        a.SetVisibility(True)
    plotter.render()

def hide_all(state):
    for a in actors.values():
        a.SetVisibility(False)
    plotter.render()

plotter.add_checkbox_button_widget(
    show_all,
    value=False,
    position=(10, int(plotter.window_size[1]*0.05)),
    size=20,
    color_on=(0.3, 0.8, 0.3),
    color_off=(0.7, 0.7, 0.7),
    border_size=1
)
plotter.add_text(
    " Show All",
    position=(35, int(plotter.window_size[1]*0.05)+3),
    font_size=8,
    color="black"
)

plotter.add_checkbox_button_widget(
    hide_all,
    value=False,
    position=(120, int(plotter.window_size[1]*0.05)),
    size=20,
    color_on=(0.8, 0.3, 0.3),
    color_off=(0.7, 0.7, 0.7),
    border_size=1
```

```
)
plotter.add_text(
    " Hide All",
    position=(145, int(plotter.window_size[1]*0.05)+3),
    font_size=8,
    color="black"
)

# ============================
# SHOW
# ============================
plotter.show_grid()
plotter.camera_position = 'xy'
plotter.show()
```

#kidney : (from TotalSegmentator)

```
import os
import nibabel as nib
import numpy as np
from skimage import measure
import pyvista as pv

# ============================
# PATH TO YOUR FOLDER
# ============================
base_dir = r"C:\Users\Asus\Downloads\segmentations (6)"

# ============================
# KIDNEY FILES (2 kidneys)
# ============================
kidney_files = {
    "kidney_right.nii.gz": "Right Kidney",
    "kidney_left.nii.gz": "Left Kidney",
}

# ============================
# COLORS
# ============================
colors = {
    "kidney_right.nii.gz": (0.8, 0.2, 0.2),  # Red
    "kidney_left.nii.gz": (0.2, 0.4, 0.9),  # Blue
}

# ============================
# CREATE PLOTTER
# ============================
plotter = pv.Plotter()
plotter.add_text("Kidney Segmentation — Left & Right", font_size=14)
plotter.add_axes(line_width=1)
plotter.add_bounding_box(color="black")

actors = {}
volumes = {}

# ============================
# LOAD EACH KIDNEY & CREATE MESH
# ============================
for fname, label in kidney_files.items():
    path = os.path.join(base_dir, fname)

    if not os.path.exists(path):
        print(f"⚠ File not found: {fname}")
        continue

    # --- Load NIfTI ---
    print(f"Loading {fname}...")
    nii = nib.load(path)
    mask = nii.get_fdata()
    mask = (mask > 0).astype(np.uint8)

    # --- Calculate volume ---
    volume_voxels = np.sum(mask)
    voxel_dims = nii.header.get_zooms()
    voxel_volume = np.prod(voxel_dims)
    volume_cm3 = (volume_voxels * voxel_volume) / 1000
    volumes[fname] = volume_cm3

    # --- Generate surface mesh using marching cubes ---
    try:
        verts, faces, _, _ = measure.marching_cubes(mask, level=0.5)
        print(f"  {label}: {len(verts)} vertices, {len(faces)} faces, Volume: {volume_cm3:.1f} cm³")
    except ValueError:
        print(f"⚠ Skipping {label} — empty or invalid mask.")
        continue

    # --- Convert to world coordinates (correct orientation) ---
    verts = nib.affines.apply_affine(nii.affine, verts)

    # --- Build mesh ---
    faces = np.hstack([np.full((faces.shape[0], 1), 3), faces]).astype(np.int32)
    mesh = pv.PolyData(verts, faces)

    # --- Add mesh to plotter ---
    actor = plotter.add_mesh(
        mesh,
        color=colors[fname],
        opacity=0.7,
        smooth_shading=True,
        name=label
    )
    actors[fname] = actor

# ============================
# ADD LEGEND
# ============================
legend_entries = [(label, colors[fname]) for fname, label in kidney_files.items() if fname in 
actors]
plotter.add_legend(legend_entries, bcolor="white", face="circle", size=(0.2, 0.15))


# ============================
# OPACITY SLIDER
# ============================
def set_opacity(val):
    val = float(val)
    for a in actors.values():
        a.GetProperty().SetOpacity(val)
    plotter.render()


plotter.add_slider_widget(
    callback=set_opacity,
    rng=[0.1, 1.0],
    value=0.7,
    title="Opacity",
    pointa=(0.02, 0.08),
    pointb=(0.35, 0.08)
)

# ============================
# CHECKBOXES TO TOGGLE KIDNEYS
# ============================
y0 = 0.80
dy = 0.08


def make_toggle(fname):
    def _cb(state):
        if fname in actors:
            actors[fname].SetVisibility(bool(state))
            plotter.render()
    return _cb


for i, (fname, label) in enumerate(kidney_files.items()):
    if fname in actors:
        plotter.add_checkbox_button_widget(
            make_toggle(fname),
            value=True,
            position=(10, int(plotter.window_size[1] * (y0 - i * dy))),
            size=25,
            color_on=colors[fname],
            color_off=(0.7, 0.7, 0.7),
            border_size=1
        )
        plotter.add_text(
            f" {label}",
            position=(40, int(plotter.window_size[1] * (y0 - i * dy)) + 5),
            font_size=10,
            color="black"
        )

# ============================
# VOLUME STATISTICS
# ============================
if volumes:
    stats_lines = ["Volume Statistics:"]
    for fname, label in kidney_files.items():
        if fname in volumes:
            stats_lines.append(f"{label}: {volumes[fname]:.1f} cm³")

    if len(volumes) == 2:
        total = sum(volumes.values())
        stats_lines.append(f"Total: {total:.1f} cm³")

    stats_text = "\n".join(stats_lines)
    plotter.add_text(
        stats_text,
        position=(10, 50),
        font_size=9,
        color="black"
    )


# ============================
# SHOW ALL / HIDE ALL BUTTONS
# ============================
def show_all(state):
    for a in actors.values():
        a.SetVisibility(True)
    plotter.render()


def hide_all(state):
    for a in actors.values():
        a.SetVisibility(False)
    plotter.render()


plotter.add_checkbox_button_widget(
    show_all,
    value=False,
    position=(10, int(plotter.window_size[1] * 0.15)),
    size=20,
    color_on=(0.3, 0.8, 0.3),
    color_off=(0.7, 0.7, 0.7),
    border_size=1
)
plotter.add_text(
    " Show All",
    position=(35, int(plotter.window_size[1] * 0.15) + 3),
    font_size=8,
    color="black"
)

plotter.add_checkbox_button_widget(
    hide_all,
    value=False,
    position=(120, int(plotter.window_size[1] * 0.15)),
    size=20,
    color_on=(0.8, 0.3, 0.3),
    color_off=(0.7, 0.7, 0.7),
    border_size=1
)
plotter.add_text(
    " Hide All",
    position=(145, int(plotter.window_size[1] * 0.15) + 3),
    font_size=8,
    color="black"
)

# ===========================
# SHOW
# ===========================
plotter.show_grid()
plotter.camera_position = 'xz'  # Better view for kidneys (front view)
plotter.show()
```

#Kidney :(MedSAM)

```
import os
import sys
import numpy as np
import nibabel as nib
import cv2
from pathlib import Path
from scipy.spatial.distance import directed_hausdorff

# Check for required libraries
try:
    from skimage import measure
    from scipy.ndimage import gaussian_filter
    VOLUME_RENDERING_AVAILABLE = True
except ImportError:
    VOLUME_RENDERING_AVAILABLE = False

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import warnings
warnings.filterwarnings('ignore')

#
==============================================================================
==
# USER CONTROLS
#
==============================================================================
==

# How many samples to process (CHANGE THIS - default is 1)
NUM_SAMPLES = 1

SHOW_CORTEX = True
SHOW_MEDULLA = True
SHOW_PELVIS = True

CORTEX_COLOR = '#FF5050'
MEDULLA_COLOR = '#50FF50'
PELVIS_COLOR = '#5050FF'

CORTEX_OPACITY = 0.7
MEDULLA_OPACITY = 0.8
PELVIS_OPACITY = 0.9
```

```
USE_VOLUME = True
DETAIL_LEVEL = 2
SMOOTH_SURFACES = True

OVERLAY_TRANSPARENCY = 0.5

VIEW_ELEVATION = 30
VIEW_ROTATION = 45

#
==========================================================================
==
# EVALUATION METRICS
#
==========================================================================
==

def dice_coefficient(pred, gt):
    pred = pred.astype(bool)
    gt = gt.astype(bool)
    intersection = np.sum(pred & gt)
    if np.sum(pred) + np.sum(gt) == 0:
        return 1.0 if intersection == 0 else 0.0
    return 2.0 * intersection / (np.sum(pred) + np.sum(gt))

def iou_score(pred, gt):
    pred = pred.astype(bool)
    gt = gt.astype(bool)
    intersection = np.sum(pred & gt)
    union = np.sum(pred | gt)
    if union == 0:
        return 1.0 if intersection == 0 else 0.0
    return intersection / union

def hausdorff_distance(pred, gt):
    pred = pred.astype(bool)
    gt = gt.astype(bool)
    pred_points = np.argwhere(pred)
    gt_points = np.argwhere(gt)
    if len(pred_points) == 0 or len(gt_points) == 0:
        return np.inf
    forward = directed_hausdorff(pred_points, gt_points)[0]
    backward = directed_hausdorff(gt_points, pred_points)[0]
    return max(forward, backward)

def calculate_metrics(pred, gt):
    return {
        'dice': dice_coefficient(pred, gt),
        'iou': iou_score(pred, gt),
        'hausdorff': hausdorff_distance(pred, gt)
    }

#
=========================================================================
==

print("\n" + "="*60)
print("KIDNEY 3-PART SEGMENTATION")
print("="*60)
print(f"Processing: {NUM_SAMPLES} sample(s)")
print(f"3D Mode: {'VOLUME' if USE_VOLUME and VOLUME_RENDERING_AVAILABLE
else 'POINT CLOUD'}")
print("="*60 + "\n")

# Paths
REPO_DIR = Path(r"C:\Users\hp\MedSAM")
possible_paths = [
    Path(r"C:\Users\hp\MedSAM\datafolder\Healthy_Control"),
    Path(r"C:\Users\hp\MedSAM\Healthy_Control"),
    Path(r"C:\Users\hp\MedSAM\MedSAM\Healthy_Control"),
    Path(r"C:\Users\hp\Downloads\Healthy_Control"),
]

DATA_PATH = None
for path in possible_paths:
    if path.exists():
        DATA_PATH = path
        break

if DATA_PATH is None:
    print("ERROR: Could not find data!")
    sys.exit(1)

OUTPUT_DIR = REPO_DIR / "kidney_healthy_results"
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

# Find files
all_nii = list(DATA_PATH.rglob(".nii"))
image_files = []
mask_files = []

for nii_file in all_nii:
    if nii_file.name.startswith('._') or nii_file.stat().st_size == 0:
        continue
    if any(k in nii_file.name.lower() for k in ['mask', 'label', 'seg', 'gt']):
        mask_files.append(nii_file)
    else:
        image_files.append(nii_file)

image_files = sorted(image_files)
mask_files = sorted(mask_files)

def load_nifti_slice(nifti_path, slice_idx=None):
    try:
        nifti_img = nib.load(str(nifti_path))
        data = nifti_img.get_fdata()
        if len(data.shape) == 2:
            slice_2d = data
            slice_idx_out = 0
            total_slices = 1
        else:
            if slice_idx is None:
                slice_idx = data.shape[2] // 2
            slice_2d = data[:, :, slice_idx]
            slice_idx_out = slice_idx
            total_slices = data.shape[2]
        slice_normalized = ((slice_2d - slice_2d.min()) /
                          (slice_2d.max() - slice_2d.min() + 1e-8) * 255).astype(np.uint8)
        return slice_normalized, slice_idx_out, total_slices
    except:
        return None, None, None

def load_nifti_volume(nifti_path):
    try:
        nifti_img = nib.load(str(nifti_path))
        return nifti_img.get_fdata()
    except:
        return None

def get_bbox_from_mask(mask_slice, margin=20):
    mask_binary = (mask_slice > 0).astype(np.uint8) * 255
    contours, _ = cv2.findContours(mask_binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) == 0:
        return None
    largest = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(largest)
    x = max(0, x - margin)
    y = max(0, y - margin)
    w = min(mask_slice.shape[1] - x, w + 2*margin)
    h = min(mask_slice.shape[0] - y, h + 2*margin)
    return np.array([x, y, x+w, y+h])

def segment_kidney_3_parts(image, full_kidney_mask):
    """Segment kidney into 3 parts based on intensity"""
    kidney_pixels = image[full_kidney_mask > 0]
    if len(kidney_pixels) == 0:
        return np.zeros_like(full_kidney_mask), np.zeros_like(full_kidney_mask),
np.zeros_like(full_kidney_mask)

    low_thresh = np.percentile(kidney_pixels, 33)
    high_thresh = np.percentile(kidney_pixels, 66)

    part1 = np.zeros_like(full_kidney_mask)
    part2 = np.zeros_like(full_kidney_mask)
    part3 = np.zeros_like(full_kidney_mask)

    part1[(full_kidney_mask > 0) & (image <= low_thresh)] = 1
    part2[(full_kidney_mask > 0) & (image > low_thresh) & (image <= high_thresh)] = 1
    part3[(full_kidney_mask > 0) & (image > high_thresh)] = 1

    return part1, part2, part3

def segment_kidney_3_parts_3d(image_volume, mask_volume):
    """3D version of segmentation"""
    kidney_pixels = image_volume[mask_volume > 0]
    if len(kidney_pixels) == 0:
        return np.zeros_like(mask_volume), np.zeros_like(mask_volume),
np.zeros_like(mask_volume)

    low_thresh = np.percentile(kidney_pixels, 33)
    high_thresh = np.percentile(kidney_pixels, 66)

    part1 = np.zeros_like(mask_volume)
    part2 = np.zeros_like(mask_volume)
    part3 = np.zeros_like(mask_volume)

    part1[(mask_volume > 0) & (image_volume <= low_thresh)] = 1
    part2[(mask_volume > 0) & (image_volume > low_thresh) & (image_volume <=
high_thresh)] = 1
    part3[(mask_volume > 0) & (image_volume > high_thresh)] = 1

    return part1, part2, part3

def hex_to_rgb(hex_color):
    hex_color = hex_color.lstrip('#')
    return tuple(int(hex_color[i:i+2], 16) for i in (0, 2, 4))

def hex_to_rgb_norm(hex_color):
    r, g, b = hex_to_rgb(hex_color)
    return (r/255.0, g/255.0, b/255.0)
```

```
def create_colored_overlay(part1, part2, part3):
    h, w = part1.shape
    colored = np.zeros((h, w, 3), dtype=np.uint8)
    if SHOW_CORTEX:
        colored[part1 > 0] = hex_to_rgb(CORTEX_COLOR)
    if SHOW_MEDULLA:
        colored[part2 > 0] = hex_to_rgb(MEDULLA_COLOR)
    if SHOW_PELVIS:
        colored[part3 > 0] = hex_to_rgb(PELVIS_COLOR)
    return colored

def create_volume_mesh(volume, spacing, smooth):
    if not VOLUME_RENDERING_AVAILABLE or np.sum(volume) == 0:
        return None, None
    try:
        if smooth:
            vol_smooth = gaussian_filter(volume.astype(float), sigma=1.5)
            vol_smooth = (vol_smooth > 0.3).astype(np.uint8)
        else:
            vol_smooth = volume
        vol_down = vol_smooth[::spacing, ::spacing, ::spacing]
        if vol_down.sum() == 0:
            return None, None
        verts, faces, _, _ = measure.marching_cubes(vol_down, level=0.5, step_size=1)
        verts = verts * spacing
        return verts, faces
    except:
        return None, None

def create_3d_visualization(part1_3d, part2_3d, part3_3d, sample_name, output_path):
    fig = plt.figure(figsize=(24, 20))
    angles = [
        (VIEW_ELEVATION, VIEW_ROTATION, "View 1"),
        (VIEW_ELEVATION, VIEW_ROTATION + 90, "View 2"),
        (VIEW_ELEVATION, VIEW_ROTATION + 180, "View 3"),
        (70, VIEW_ROTATION, "Top View")
    ]

    if USE_VOLUME and VOLUME_RENDERING_AVAILABLE:
        meshes = []
        if SHOW_CORTEX:
            v, f = create_volume_mesh(part1_3d, DETAIL_LEVEL, SMOOTH_SURFACES)
            if v is not None:
                meshes.append(('Cortex', v, f, CORTEX_COLOR, CORTEX_OPACITY))
        if SHOW_MEDULLA:
            v, f = create_volume_mesh(part2_3d, DETAIL_LEVEL, SMOOTH_SURFACES)
            if v is not None:
                meshes.append(('Medulla', v, f, MEDULLA_COLOR, MEDULLA_OPACITY))
        if SHOW_PELVIS:
            v, f = create_volume_mesh(part3_3d, DETAIL_LEVEL, SMOOTH_SURFACES)
            if v is not None:
                meshes.append(('Pelvis', v, f, PELVIS_COLOR, PELVIS_OPACITY))

        if not meshes:
            plt.close()
            return

        for idx, (elev, azim, title) in enumerate(angles, 1):
            ax = fig.add_subplot(2, 2, idx, projection='3d')
            for name, verts, faces, color, alpha in meshes:
                mesh = Poly3DCollection(verts[faces], alpha=alpha, linewidth=0.05)
                mesh.set_facecolor(hex_to_rgb_norm(color))
                mesh.set_edgecolor((0, 0, 0, 0.1))
                ax.add_collection3d(mesh)
            ax.set_xlabel('X', fontsize=12)
            ax.set_ylabel('Y', fontsize=12)
            ax.set_zlabel('Z', fontsize=12)
            ax.set_title(title, fontsize=16)
            ax.view_init(elev=elev, azim=azim)
            ax.set_facecolor('#f5f5f5')
            all_v = np.vstack([m[1] for m in meshes])
            max_range = np.array([
                all_v[:, 0].max() - all_v[:, 0].min(),
                all_v[:, 1].max() - all_v[:, 1].min(),
                all_v[:, 2].max() - all_v[:, 2].min()
            ]).max() / 2.0
            mid = [(all_v[:, i].max() + all_v[:, i].min()) / 2 for i in range(3)]
            ax.set_xlim(mid[0] - max_range, mid[0] + max_range)
            ax.set_ylim(mid[1] - max_range, mid[1] + max_range)
            ax.set_zlim(mid[2] - max_range, mid[2] + max_range)
    else:
        ds = 2
        p1 = part1_3d[::ds, ::ds, ::ds]
        p2 = part2_3d[::ds, ::ds, ::ds]
        p3 = part3_3d[::ds, ::ds, ::ds]
        x1, y1, z1 = np.where(p1 > 0) if SHOW_CORTEX else ([], [], [])
        x2, y2, z2 = np.where(p2 > 0) if SHOW_MEDULLA else ([], [], [])
        x3, y3, z3 = np.where(p3 > 0) if SHOW_PELVIS else ([], [], [])
        for idx, (elev, azim, title) in enumerate(angles, 1):
            ax = fig.add_subplot(2, 2, idx, projection='3d')
            if SHOW_CORTEX and len(x1) > 0:
                ax.scatter(x1, y1, z1, c=CORTEX_COLOR, s=12, alpha=CORTEX_OPACITY)
            if SHOW_MEDULLA and len(x2) > 0:
                ax.scatter(x2, y2, z2, c=MEDULLA_COLOR, s=12, alpha=MEDULLA_OPACITY)
            if SHOW_PELVIS and len(x3) > 0:
                ax.scatter(x3, y3, z3, c=PELVIS_COLOR, s=12, alpha=PELVIS_OPACITY)
            ax.set_title(title, fontsize=16)
            ax.view_init(elev=elev, azim=azim)

    plt.suptitle(f'3D Kidney: {sample_name}', fontsize=22, y=0.98)
    plt.tight_layout()
    plt.savefig(output_path, dpi=150, bbox_inches='tight')
    plt.close()

def match_mask_to_image(img_name, mask_files):
    img_base = img_name.replace('.nii.gz', '').replace('.nii', '')
    for mask_file in mask_files:
        mask_base = mask_file.stem.replace('.nii', '')
        if img_base in mask_base or mask_base in img_base:
            return mask_file
    return None

#
========================================================================
==
# MAIN PROCESSING
#
========================================================================
==

print("Processing samples...\n")

results = []
all_metrics = []

for i in range(min(NUM_SAMPLES, len(image_files))):
    img_path = image_files[i]
    print(f"[{i+1}/{NUM_SAMPLES}] {img_path.name}")

    mask_path = match_mask_to_image(img_path.name, mask_files)
    if not mask_path:
        print("  ✗ No matching mask found\n")
        continue

    # Load 2D slice
    img_slice, slice_idx, num_slices = load_nifti_slice(img_path)
    if img_slice is None:
        print("  ✗ Failed to load image\n")
        continue

    mask_slice, _, _ = load_nifti_slice(mask_path, slice_idx)
    if mask_slice is None:
        print("  ✗ Failed to load mask\n")
        continue
    # Prepare data
    kidney_mask = (mask_slice > 0).astype(np.uint8)
    bbox = get_bbox_from_mask(kidney_mask)
    img_resized = cv2.resize(img_slice, (1024, 1024))
    mask_resized = cv2.resize(kidney_mask, (1024, 1024),
interpolation=cv2.INTER_NEAREST)
    bbox_resized = (bbox * (1024 / img_slice.shape[0])).astype(int) if bbox is not None else
None

    if len(img_resized.shape) == 2:
        img_3channel = cv2.cvtColor(img_resized, cv2.COLOR_GRAY2RGB)
    else:
        img_3channel = img_resized

    # 2D Segmentation - SEGMENT THE IMAGE, NOT THE MASK
    gray_img = cv2.cvtColor(img_3channel, cv2.COLOR_RGB2GRAY) if
len(img_3channel.shape) == 3 else img_3channel

    # First, detect kidney region in the image (not using mask for segmentation)
    # Apply adaptive thresholding to find kidney region
    blurred = cv2.GaussianBlur(gray_img, (5, 5), 0)
    _, binary = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    # Find largest connected component (kidney)
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) == 0:
        print("  ✗ No kidney detected in image\n")
        continue

    # Create kidney mask from detected region
    kidney_detected = np.zeros_like(gray_img, dtype=np.uint8)
    largest_contour = max(contours, key=cv2.contourArea)
    cv2.fillPoly(kidney_detected, [largest_contour], 255)
    kidney_detected = (kidney_detected > 0).astype(np.uint8)

    # Now segment the detected kidney into 3 parts based on intensity
    part1_2d, part2_2d, part3_2d = segment_kidney_3_parts(gray_img, kidney_detected)

    # Calculate metrics - Compare segmented kidney vs ground truth mask
    combined_pred = (part1_2d + part2_2d + part3_2d > 0).astype(np.uint8)
    metrics = calculate_metrics(combined_pred, mask_resized)  # mask_resized is ground
truth
    all_metrics.append({'sample': img_path.stem, **metrics})

    print(f"  Detection: {np.sum(combined_pred)} px | Ground Truth: {np.sum(mask_resized)}
px")
```

```
    print(f"  Dice: {metrics['dice']:.3f} | IoU: {metrics['iou']:.3f} | HD: {metrics['hausdorff']:.1f}px")

    colored_2d = create_colored_overlay(part1_2d, part2_2d, part3_2d)

    # Create 2D visualization
    fig, axes = plt.subplots(2, 3, figsize=(20, 13))
    fig.suptitle(f"Kidney 3-Part Segmentation: {img_path.stem}\nDice={metrics['dice']:.3f} |
IoU={metrics['iou']:.3f} | HD={metrics['hausdorff']:.1f}px",
                 fontsize=18, y=0.98)

    axes[0, 0].imshow(img_3channel, cmap='gray')
    if bbox_resized is not None:
        rect = plt.Rectangle((bbox_resized[0], bbox_resized[1]),
                             bbox_resized[2] - bbox_resized[0], bbox_resized[3] - bbox_resized[1],
                             fill=False, edgecolor='yellow', linewidth=3)
        axes[0, 0].add_patch(rect)
    axes[0, 0].set_title('Original + BBox', fontsize=14)
    axes[0, 0].axis('off')

    axes[0, 1].imshow(img_3channel, cmap='gray')
    axes[0, 1].imshow(colored_2d, alpha=OVERLAY_TRANSPARENCY)
    axes[0, 1].set_title('Detected 3-Part Segmentation', fontsize=14)
    axes[0, 1].axis('off')

    axes[0, 2].imshow(img_3channel, cmap='gray')
    gt_colored = np.zeros_like(colored_2d)
    gt_colored[mask_resized > 0] = [255, 200, 0]
    axes[0, 2].imshow(gt_colored, alpha=0.5)
    axes[0, 2].set_title('Ground Truth Mask', fontsize=14)
    axes[0, 2].axis('off')

    axes[1, 0].imshow(part1_2d, cmap='Reds', vmin=0, vmax=1)
    axes[1, 0].set_title(f'Part 1 (Low Intensity)\n{np.sum(part1_2d)} px', fontsize=12)
    axes[1, 0].axis('off')

    axes[1, 1].imshow(part2_2d, cmap='Greens', vmin=0, vmax=1)
    axes[1, 1].set_title(f'Part 2 (Mid Intensity)\n{np.sum(part2_2d)} px', fontsize=12)
    axes[1, 1].axis('off')

    axes[1, 2].imshow(part3_2d, cmap='Blues', vmin=0, vmax=1)
    axes[1, 2].set_title(f'Part 3 (High Intensity)\n{np.sum(part3_2d)} px', fontsize=12)
    axes[1, 2].axis('off')

    plt.tight_layout()
    viz_2d_path = OUTPUT_DIR / f"2D_{i+1}_{img_path.stem}.png"
    plt.savefig(viz_2d_path, dpi=150, bbox_inches='tight')
    plt.close()
    # 3D processing - Segment the IMAGE volume, not the mask
    img_volume = load_nifti_volume(img_path)
    mask_volume = load_nifti_volume(mask_path)

    if img_volume is not None and mask_volume is not None:
        img_volume_norm = ((img_volume - img_volume.min()) /
                          (img_volume.max() - img_volume.min() + 1e-8) * 255).astype(np.uint8)
        mask_volume_binary = (mask_volume > 0).astype(np.uint8)

        # Detect kidney in 3D volume using thresholding
        blurred_vol = gaussian_filter(img_volume_norm.astype(float), sigma=1.0) if
VOLUME_RENDERING_AVAILABLE else img_volume_norm
        threshold_val = np.percentile(img_volume_norm[img_volume_norm > 0], 30)
        kidney_detected_3d = (blurred_vol > threshold_val).astype(np.uint8)

        # Segment detected kidney into 3 parts
        part1_3d, part2_3d, part3_3d = segment_kidney_3_parts_3d(img_volume_norm,
kidney_detected_3d)

        viz_3d_path = OUTPUT_DIR / f"3D_{i+1}_{img_path.stem}.png"
        create_3d_visualization(part1_3d, part2_3d, part3_3d, img_path.stem, viz_3d_path)
        results.append({'name': img_path.stem, '2d_path': viz_2d_path, '3d_path':
viz_3d_path})
    else:
        results.append({'name': img_path.stem, '2d_path': viz_2d_path, '3d_path': None})

    print(f"  ✓ Saved to {OUTPUT_DIR}\n")

# Save metrics
if all_metrics:
    import csv
    metrics_csv = OUTPUT_DIR / "evaluation_metrics.csv"
    with open(metrics_csv, 'w', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=['sample', 'dice', 'iou', 'hausdorff'])
        writer.writeheader()
        writer.writerows(all_metrics)

    print(f"\n{'='*60}")
    print("SUMMARY")
    print(f"{'='*60}")
    avg_dice = np.mean([m['dice'] for m in all_metrics])
    avg_iou = np.mean([m['iou'] for m in all_metrics])
    avg_hd = np.mean([m['hausdorff'] for m in all_metrics])
    print(f"Average Dice:      {avg_dice:.4f}")
    print(f"Average IoU:       {avg_iou:.4f}")
    print(f"Average Hausdorff: {avg_hd:.2f} px")
    print(f"\nResults saved to: {OUTPUT_DIR}")
    print(f"{'='*60}\n")
```

```
# Open only the FIRST result
if results:
    try:
        os.startfile(str(results[0]['2d_path']))
        if results[0].get('3d_path'):
            os.startfile(str(results[0]['3d_path']))
    except:
        print("Note: Could not auto-open images (this is normal on some systems)")
```

#Liver (MedSam)

```
import os
import pickle
import json
import numpy as np
import nibabel as nib
import pyvista as pv
from scipy import ndimage
from skimage import measure
from tqdm import tqdm


class MultiPartLiver3DVisualizer:
    """3D visualization of 3-part liver segmentation with transparency and colors"""

    def _init_(self):
        self.part_names = {
            1: 'Left Lobe',
            2: 'Right Lobe',
            3: 'Caudate Lobe'
        }

        self.part_colors = {
            1: '#FF6B6B',  # Red
            2: '#4ECDC4',  # Cyan/Turquoise
            3: '#FFE66D'  # Yellow
        }

        self.meshes = {}
        self.actors = {}
        self.visibility = {1: True, 2: True, 3: True}

    def load_3part_segmentation(self, file_path):
        """Load 3-part segmentation from saved file"""
        print(f"Loading 3-part segmentation: {file_path}")

        if file_path.endswith('.pkl'):
            with open(file_path, 'rb') as f:
                data = pickle.load(f)
            segmentation = data['segmentation']
            stats = data.get('stats', {})
            affine = data['affine']

        elif file_path.endswith('.npz'):
            data = np.load(file_path)
            segmentation = data['segmentation']
            affine = data['affine']

            json_path = file_path.replace('.npz', '_metadata.json')
            stats = {}
            if os.path.exists(json_path):
                with open(json_path, 'r') as f:
                    stats = json.load(f)

        elif file_path.endswith('.nii.gz') or file_path.endswith('.nii'):
            nii = nib.load(file_path)
            segmentation = nii.get_fdata().astype(np.uint8)
            affine = nii.affine

            json_path = file_path.replace('.nii.gz', '_metadata.json').replace('.nii',
'_metadata.json')
            stats = {}
            if os.path.exists(json_path):
                with open(json_path, 'r') as f:
                    stats = json.load(f)
        else:
            raise ValueError(f"Unsupported file format: {file_path}")

        unique_labels = np.unique(segmentation)
        print(f"✓ Loaded segmentation: {segmentation.shape}")
        print(f"  Unique labels: {unique_labels}")

        for part_id in [1, 2, 3]:
            count = (segmentation == part_id).sum()
            print(f"  {self.part_names[part_id]}: {count:,} voxels")

        return segmentation, stats, affine

    def create_mesh_for_part(self, segmentation_volume, part_id, voxel_spacing=(1.0, 1.0,
1.0)):
        """Create mesh for a specific liver part"""
        print(f"Creating mesh for {self.part_names[part_id]}...")

        # Extract part volume
        part_volume = (segmentation_volume == part_id).astype(np.float32)
```

```
voxel_count = (part_volume > 0).sum()

print(f"  Voxels: {voxel_count:,}")

if voxel_count < 100:
    print(f"  ⚠ Too few voxels, skipping")
    return None

try:
    # Light preprocessing
    part_binary = part_volume > 0

    # Fill holes
    part_binary = ndimage.binary_fill_holes(part_binary)

    # Keep largest component
    labeled, num_features = ndimage.label(part_binary)
    if num_features > 1:
        component_sizes = np.bincount(labeled.ravel())
        component_sizes[0] = 0
        largest = component_sizes.argmax()
        part_binary = (labeled == largest)

    # Very light smoothing
    if voxel_count > 1000:
        part_binary = ndimage.binary_closing(part_binary, iterations=1)

    part_volume = part_binary.astype(np.float32)

    # Marching cubes
    level = 0.5
    verts, faces, normals, values = measure.marching_cubes(
        part_volume,
        level=level,
        spacing=voxel_spacing,
        allow_degenerate=False
    )

    print(f"  Mesh: {len(verts):,} vertices, {len(faces):,} faces")

    # Create PyVista mesh
    faces_pv = np.hstack([[3] + face.tolist() for face in faces])
    mesh = pv.PolyData(verts, faces_pv)

    # Clean and smooth
    mesh = mesh.clean()

    # Adaptive smoothing based on size
```

```
            smooth_iter = min(50, max(10, int(voxel_count / 100)))
            mesh = mesh.smooth(n_iter=smooth_iter, relaxation_factor=0.15)

            # Compute normals
            mesh = mesh.compute_normals(cell_normals=True, point_normals=True)

            print(f"  ✓ Final: {mesh.n_points:,} points")

            return mesh

        except Exception as e:
            print(f"  ✗ Error: {e}")
            return None

    def create_all_meshes(self, segmentation_volume, voxel_spacing=(1.0, 1.0, 1.0)):
        """Create meshes for all 3 parts"""
        print("\nCreating 3D meshes for all parts...")
        print("=" * 60)

        for part_id in [1, 2, 3]:
            mesh = self.create_mesh_for_part(segmentation_volume, part_id, voxel_spacing)
            if mesh is not None:
                self.meshes[part_id] = mesh

        print("=" * 60)
        print(f"✓ Created {len(self.meshes)} meshes")

        if len(self.meshes) == 0:
            raise ValueError("No meshes created! Check your segmentation data.")

    def visualize_interactive(self, segmentation_volume, stats=None,
                              voxel_spacing=(1.0, 1.0, 1.0),
                              opacity=0.7, window_size=(1400, 1000)):
        """
        Interactive 3D visualization with transparent colored parts

        Controls:
        - Press '1' to toggle Left Lobe
        - Press '2' to toggle Right Lobe
        - Press '3' to toggle Caudate Lobe
        - Press 'o' to cycle opacity (0.3, 0.5, 0.7, 0.9)
        - Press 'r' to reset camera
        - Press 'q' to quit
        """
        # Create meshes
        if not self.meshes:
            self.create_all_meshes(segmentation_volume, voxel_spacing)
```

```
if not self.meshes:
    raise ValueError("No meshes to display!")

# Create plotter
plotter = pv.Plotter(window_size=window_size)
plotter.set_background('white')

# Store opacity for cycling
self.current_opacity = opacity

# Add each part with transparency
for part_id, mesh in self.meshes.items():
    actor = plotter.add_mesh(
        mesh,
        color=self.part_colors[part_id],
        opacity=opacity,
        smooth_shading=True,
        specular=0.3,
        specular_power=10,
        label=self.part_names[part_id]
    )
    self.actors[part_id] = actor

# Add legend
plotter.add_legend(bcolor='white', face='circle', size=(0.2, 0.2))

# Add axes with anatomical labels
plotter.add_axes(
    xlabel='Right → Left',
    ylabel='Anterior → Posterior',
    zlabel='Inferior → Superior',
    line_width=3
)

# Set camera
plotter.camera_position = 'iso'
plotter.camera.zoom(1.3)

# Add lighting
light1 = pv.Light(position=(2, 2, 2), light_type='camera light')
light2 = pv.Light(position=(-2, -2, 2), light_type='camera light', intensity=0.5)
plotter.add_light(light1)
plotter.add_light(light2)

# Title
title = "3D Multi-Part Liver Segmentation\n"
title += "Press 1/2/3 to toggle parts | O: Opacity | R: Reset | Q: Quit"
plotter.add_title(title, font_size=12, color='black')
```

```
        # Stats text
        if stats:
            info_text = f"Left Lobe: {stats.get('left_lobe_ml', 0):.1f} mL\n"
            info_text += f"Right Lobe: {stats.get('right_lobe_ml', 0):.1f} mL\n"
            info_text += f"Caudate: {stats.get('caudate_lobe_ml', 0):.1f} mL\n"
            info_text += f"Total: {stats.get('total_ml', 0):.1f} mL\n"
            info_text += f"Opacity: {opacity:.1f}"

            self.stats_actor = plotter.add_text(
                info_text,
                position='lower_right',
                font_size=10,
                color='black'
            )

        # Key callbacks
        def toggle_part(part_id):
            if part_id in self.actors:
                self.visibility[part_id] = not self.visibility[part_id]
                self.actors[part_id].SetVisibility(self.visibility[part_id])
                status = "ON" if self.visibility[part_id] else "OFF"
                print(f"{self.part_names[part_id]}: {status}")

        def cycle_opacity():
            opacities = [0.3, 0.5, 0.7, 0.9]
            current_idx = opacities.index(min(opacities, key=lambda x: abs(x -
self.current_opacity)))
            next_idx = (current_idx + 1) % len(opacities)
            self.current_opacity = opacities[next_idx]

            for actor in self.actors.values():
                actor.GetProperty().SetOpacity(self.current_opacity)

            # Update stats text
            if stats and hasattr(self, 'stats_actor'):
                info_text = f"Left Lobe: {stats.get('left_lobe_ml', 0):.1f} mL\n"
                info_text += f"Right Lobe: {stats.get('right_lobe_ml', 0):.1f} mL\n"
                info_text += f"Caudate: {stats.get('caudate_lobe_ml', 0):.1f} mL\n"
                info_text += f"Total: {stats.get('total_ml', 0):.1f} mL\n"
                info_text += f"Opacity: {self.current_opacity:.1f}"
                self.stats_actor.SetText(3, info_text)

            print(f"Opacity: {self.current_opacity:.1f}")

        plotter.add_key_event('1', lambda: toggle_part(1))
        plotter.add_key_event('2', lambda: toggle_part(2))
        plotter.add_key_event('3', lambda: toggle_part(3))
        plotter.add_key_event('o', cycle_opacity)
        plotter.add_key_event('O', cycle_opacity)

        # Show instructions
        print("\n" + "=" * 60)
        print("INTERACTIVE 3D VIEWER - CONTROLS")
        print("=" * 60)
        print("Press '1' - Toggle Left Lobe (Red)")
        print("Press '2' - Toggle Right Lobe (Cyan)")
        print("Press '3' - Toggle Caudate Lobe (Yellow)")
        print("Press 'o' - Cycle opacity (0.3 → 0.5 → 0.7 → 0.9)")
        print("Press 'r' - Reset camera")
        print("Press 'q' - Quit")
        print("Mouse: Left=Rotate, Middle=Pan, Right/Scroll=Zoom")
        print("=" * 60 + "\n")

        plotter.show()

    def save_views(self, segmentation_volume, output_dir,
                   voxel_spacing=(1.0, 1.0, 1.0), opacity=0.7):
        """Save standard anatomical views"""
        if not self.meshes:
            self.create_all_meshes(segmentation_volume, voxel_spacing)

        os.makedirs(output_dir, exist_ok=True)

        views = {
            'anterior': {'position': (0, -1, 0), 'up': (0, 0, 1)},
            'posterior': {'position': (0, 1, 0), 'up': (0, 0, 1)},
            'right_lateral': {'position': (1, 0, 0), 'up': (0, 0, 1)},
            'left_lateral': {'position': (-1, 0, 0), 'up': (0, 0, 1)},
            'superior': {'position': (0, 0, 1), 'up': (0, 1, 0)},
            'inferior': {'position': (0, 0, -1), 'up': (0, 1, 0)},
            'oblique': 'iso'
        }

        print(f"\nCreating standard views in: {output_dir}")

        for view_name, view_config in views.items():
            plotter = pv.Plotter(off_screen=True, window_size=(1200, 900))
            plotter.set_background('white')

            for part_id, mesh in self.meshes.items():
                plotter.add_mesh(
                    mesh,
                    color=self.part_colors[part_id],
                    opacity=opacity,
                    smooth_shading=True,
                    label=self.part_names[part_id]
                )

            if view_config == 'iso':
                plotter.camera_position = 'iso'
            else:
                plotter.camera.position = view_config['position']
                plotter.camera.up = view_config['up']

            plotter.camera.zoom(1.4)
            plotter.add_legend(bcolor='white')
            plotter.add_axes(line_width=3)
            plotter.add_title(f"3-Part Liver - {view_name.replace('_', ' ').title()}", font_size=16)

            output_path = os.path.join(output_dir, f'liver_3part_{view_name}.png')
            plotter.screenshot(output_path)
            plotter.close()

            print(f"  ✓ {view_name}")

        print(f"✓ Saved {len(views)} views")


def main():
    """Main function"""

    CONFIG = {
        'segmentation_file':
r'C:\Users\Hend\med-seg-3organs\pred_masks\img1_medsam_fast.nii.gz',
        'output_dir': r'C:\Users\Hend\med-seg-3organs\results_multipart\3d_views',
        'voxel_spacing': (1.0, 1.0, 1.0),
        'opacity': 0.7,
    }

    os.makedirs(CONFIG['output_dir'], exist_ok=True)

    print("=" * 60)
    print("3D MULTI-PART LIVER VISUALIZATION")
    print("=" * 60)

    # Initialize visualizer
    visualizer = MultiPartLiver3DVisualizer()

    # Load 3-part segmentation
    segmentation, stats, affine = visualizer.load_3part_segmentation(
        CONFIG['segmentation_file']
    )
    # Check if we have all 3 parts
    unique_labels = np.unique(segmentation)
    if len(unique_labels[unique_labels > 0]) < 3:
        print("\n⚠ WARNING: Less than 3 parts found in segmentation!")
        print("You need to run the 3-part segmentation first with step=1")
        print("Expected labels: 1 (Left Lobe), 2 (Right Lobe), 3 (Caudate Lobe)")
        return

    # Print stats
    if stats:
        print("\nSegmentation Statistics:")
        print("-" * 60)
        print(f"Left Lobe:    {stats.get('left_lobe_ml', 0):.2f} mL")
        print(f"Right Lobe:   {stats.get('right_lobe_ml', 0):.2f} mL")
        print(f"Caudate Lobe: {stats.get('caudate_lobe_ml', 0):.2f} mL")
        print(f"Total:        {stats.get('total_ml', 0):.2f} mL")

    # Save standard views
    print("\nCreating standard views...")
    visualizer.save_views(
        segmentation,
        CONFIG['output_dir'],
        voxel_spacing=CONFIG['voxel_spacing'],
        opacity=CONFIG['opacity']
    )

    # Interactive visualization
    print("\nLaunching interactive viewer...")
    visualizer.visualize_interactive(
        segmentation,
        stats=stats,
        voxel_spacing=CONFIG['voxel_spacing'],
        opacity=CONFIG['opacity']
    )

    print("\n✓ Complete!")


if _name_ == "_main_":
    main()
```