

## Aim

To develop a pipeline that processes an image to estimate depth and reconstruct a 3D mesh using depth estimation models and point cloud processing techniques.

## Objective

1. To utilize a pretrained depth estimation model to predict the depth map of an image.
2. To convert the depth map and corresponding RGB image into a point cloud.
3. To clean and process the point cloud for accurate 3D reconstruction.
4. To visualize and export the final 3D mesh.

## Summary

This project involves using a depth estimation model from the Hugging Face Transformers library to predict depth from a 2D image. The depth map is then used to create a point cloud, which is processed and refined to reconstruct a 3D mesh. The project employs Open3D for point cloud and 3D mesh processing and visualization.

## Tools and Libraries Used

1. **Python**: Programming language for implementing the project.
2. **Jupyter Lab**: Interactive environment for writing and running code.
3. **Hugging Face Transformers**: Library for accessing pretrained models, specifically GLPN for depth estimation.
4. **PIL (Python Imaging Library)**: For image processing.
5. **Torch (PyTorch)**: Deep learning library for tensor operations and model inference.
6. **Matplotlib**: Visualization library for plotting images and depth maps.
7. **Open3D**: Library for 3D data processing and visualization.
8. **NumPy**: Library for numerical computations.

## Procedure

1. **Importing Libraries**:
  - Import necessary libraries for image processing, model inference, and 3D data handling.

CODE:

```
import matplotlib
matplotlib.use('TkAgg')
from matplotlib import pyplot as plt
```

```
from PIL import Image
import torch
from transformers import GLPNImageProcessor, GLPNForDepthEstimation
import numpy as np
import open3d as o3d
```

## 2. Loading the Pretrained Model:

- Load the GLPN (Global-Local Path Network) model and feature extractor for depth estimation.

CODE:

```
feature_extractor = GLPNImageProcessor.from_pretrained("vinvino02/glpn-nyu")
model = GLPNForDepthEstimation.from_pretrained("vinvino02/glpn-nyu")
```

## 3. Image Preprocessing:

- Open and resize the image to suitable dimensions for the model.

CODE:

```
image = Image.open("images/gamingdesk.jpg")
new_height = 480 if image.height > 480 else image.height
new_height = new_height - (new_height % 32)
new_width = int(new_height * image.width / image.height)
diff = new_width % 32
new_width = new_width - diff if diff < 16 else new_width + 32 - diff
new_size = (new_width, new_height)
image = image.resize(new_size)
```

## 4. Model Inference:

- Prepare the image for the model and run the inference to get the predicted depth map.

CODE:

```
inputs = feature_extractor(images=image, return_tensors="pt")
with torch.no_grad():
    outputs = model(**inputs)
    predicted_depth = outputs.predicted_depth
```

```
predicted_depth = outputs.predicted_depth
```

#### 5. Post-Processing the Depth Map:

- Post-process the predicted depth map and prepare for visualization.

CODE:

```
output = predicted_depth.squeeze().cpu().numpy() * 1000.0
pad = 16
output = output[pad:-pad, pad:-pad]
image = image.crop((pad, pad, image.width - pad, image.height - pad))
```

#### 6. Visualization:

- Visualize the original image and the depth map side by side.

CODE:

```
fig, ax = plt.subplots(1, 2)
ax[0].imshow(image)
ax[0].tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)
ax[1].imshow(output, cmap='plasma')
ax[1].tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)
plt.tight_layout()
plt.pause(5)
```

#### 7. Point Cloud Creation:

- Convert the depth map and image to Open3D's RGBD image format and create a point cloud.

CODE:

```
width, height = image.size
depth_image = (output * 255 / np.max(output)).astype('uint8')
image = np.array(image)
depth_o3d = o3d.geometry.Image(depth_image)
image_o3d = o3d.geometry.Image(image)
rgb_image = o3d.geometry.RGBDImage.create_from_color_and_depth(image_o3d, depth_o3d,
convert_rgb_to_intensity=False)
```

```
camera_intrinsic = o3d.camera.PinholeCameraIntrinsic()
camera_intrinsic.set_intrinsics(width, height, 500, 500, width/2, height/2)
pcd_raw = o3d.geometry.PointCloud.create_from_rgbd_image(rgbd_image, camera_intrinsic)
```

#### 8. Point Cloud Processing:

- Remove outliers and estimate normals for the point cloud.

CODE:

```
cl, ind = pcd_raw.remove_statistical_outlier(nb_neighbors=20, std_ratio=6.0)
pcd = pcd_raw.select_by_index(ind)
pcd.estimate_normals()
pcd.orient_normals_to_align_with_direction()
```

#### 9. Surface Reconstruction:

- Reconstruct the surface to create a 3D mesh and visualize it.

CODE:

```
mesh = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(pcd, depth=10, n_threads=1)[0]
rotation = mesh.get_rotation_matrix_from_xyz((np.pi, 0, 0))
mesh.rotate(rotation, center=(0, 0, 0))
o3d.visualization.draw_geometries([mesh], mesh_show_back_face=True)
```

#### 10. Mesh Refinement and Export:

- Refine the mesh, visualize it, and export the final 3D model.

CODE:

```
mesh_uniform = mesh.paint_uniform_color([0.9, 0.8, 0.9])
mesh_uniform.compute_vertex_normals()
o3d.visualization.draw_geometries([mesh_uniform], mesh_show_back_face=True)
o3d.io.write_triangle_mesh('results/mesh4.ply', mesh)
```

### Highlights

- **Use of Pretrained Model:** Leveraged a pretrained depth estimation model (GLPN) for efficient and accurate depth prediction.
- **Image and Depth Map Visualization:** Utilized Matplotlib for clear visualization of both the original image and the depth map.

- **Open3D for 3D Data Processing:** Employed Open3D to create and process point clouds, perform surface reconstruction, and visualize 3D meshes.
- **Comprehensive Workflow:** Included steps for image preprocessing, model inference, depth map post-processing, point cloud creation, outlier removal, normal estimation, surface reconstruction, and mesh export.

## **Conclusion**

This project successfully demonstrated the use of advanced depth estimation and 3D reconstruction techniques. By integrating various tools and libraries, the pipeline efficiently processed a 2D image into a 3D model. The approach can be further expanded for more complex scenes or integrated into larger systems for applications such as 3D modeling, virtual reality, or robotic navigation.