AI-2-Print Setup Information

Written By: Phillip Goldberg

These instructions have been tested to run on Google Colab with their provided resources. These steps should work on your own machine but for best support use Google Colab.

! and **%** symbols are only needed to run commands in Colab. When changing directories on your own machine the root directory will not be **"/content"** so make sure your paths are correct.

The repository has the python files and or the Jupyter Notebook file, so you do not have to running through every step but if you want to try it yourself, I highly recommend it.

1. Get needed files

- First you need to clone Shap-E from OpenAI's repository: https://github.com/openai/shap-e https://github.com/Nehri/slicing_algorithm.git

```
!git clone https://github.com/openai/shap-e.git
```

- Then you need to clone the Slicing Algorithm from their repository:

```
!git clone https://github.com/Nehri/slicing_algorithm.git
```

2. Then you need to install Shap-E:

```
%cd /content/shap-e/
!pip install -e.
```

3. Then you need to install required python packages using pip:

```
!pip install pymeshlab # this is a library that deals with processing 3d meshes
efficiently
!pip install numpy # this is a library that deals with computation
!pip install torch # this is the library that deals with the machine learning
computation
!pip install numpy-stl # this library adds stl functionality to numPy
!pip install stlconverter # this is used to convert between ASCII and Binary STL
files
!pip install colorama # this allows color ASCII text in console and is a
dependency for the slicing-algorithm project
```

4. Now you need to import the libraries required:

```
import torch
import pymeshlab
from shap_e.diffusion.sample import sample_latents
from shap_e.diffusion.gaussian_diffusion import diffusion_from_config
from shap_e.models.download import load_model, load_config
from shap_e.util.notebooks import decode_latent_mesh
from shap_e.util.notebooks import create_pan_cameras, decode_latent_images,
gif_widget
```

5. Set torch to use your GPU instead of CPU for faster processing:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') # This line
sets torch to use a GPU for much faster processing
```

6. Set the Stable Diffusion Models:

```
xm = load_model('transmitter', device=device)
model = load_model('text300M', device=device)
diffusion = diffusion_from_config(load_config('diffusion'))
```

7. Setup Shap-E/Stable Diffusion Setting and User Prompt:

```
batch_size = 1
guidance_scale = 15, # this determines the freedom of the prompt, higher number
means more accurate to the description
prompt = "baby grand piano"  # area for user to type text prompt for mesh
generation

latents = sample_latents(
    batch_size=batch_size,
    model=model,
    diffusion=diffusion,
    guidance_scale=guidance_scale,
    model_kwargs=dict(texts=[prompt] * batch_size),
    progress=True, # shows user how far along the process is "In this case a
progress bar"
    clip_denoised=True, # this important to get rid of numbers outside range for
usable outputs
    use_fp16=True, # this sets the model to use 16-bit floating point precision
when the model does computations
    use_karras=True, # enables the model to use karras steps
    karras_steps=20, # this sets the amount of noise in processing the mesh
    sigma_min=1e-3, # this sets the minimum amount of noise the model can use when
```

```
sampling
    sigma_max=160, # this the same but for the maximum
    s_churn=0, # controls the level of variability in the sampling process which
can diversify the "creativity" of the outputs
)
```

8. Write the Generated Mesh to OBJ File:

```
# --- this code decodes latents into a mesh, and then writes the meshes data to an
OBJ file ---
for i, latent in enumerate(latents):
    t = decode_latent_mesh(xm, latent).tri_mesh()
    with open(f'mesh_{i}.obj', 'w') as f:
        t.write_obj(f)
```

9. Convert OBJ file to STL:

```
ms = pymeshlab.MeshSet()

# Load the OBJ file
ms.load_new_mesh('/content/shap-e/mesh_0.obj')

# Save as STL
ms.save_current_mesh('mesh.stl')
```

10. Convert Binary STL File to ASCII Formatted STL:

- This is required because the slicing script cannot use binary formatting

```
!python3 -m stlconverter /content/shap-e/mesh.stl stla
```

11. Slice the STL File into Gcode for 3D printing:

- Slicing STL Mesh with User Provided Parameters: 0.25 is layer height and 0 is infill percentage. It is important to mention that your printer configuration may be different from the defaults in the slicing.py file. Bed size and a few other settings can be easily changed in that file.

```
!python3 /content/slicing_algorithm/slicing.py /content/shap-e/mesh-converted-
ASCII.stl 0.25 0
```

- It is important to note that slicing 3D models generated by Shap-E will take a very long to to slice, even when running in Google Colab. This could be hours long. I highly recommend using software like Cura,

Simplfiy3D, or Slic3r to slice your generated models as it will be very fast. That way your models slice in seconds. The use of Slicing Algorithm is good at simple meshes and using it for high triangle counts is experimental. Models generated can have holes and defects that can make parts unprintable. To fixed these issues you can use tools like Blender and Meshmixer to repair meshes.