ABSTRACT

The **Art Relief Design App** is a web-based application developed to facilitate the transformation of two-dimensional images into three-dimensional relief art. Utilizing brightness-based height mapping, the system generates 3D surface models (STL files) and corresponding machine instructions (G-code) suitable for 3D printing and CNC machining. Implemented using Python and Streamlit, the application offers real-time 3D visualization, adjustable design parameters, and batch processing capabilities. Additionally, the system produces a detailed PDF report encompassing the original image, height map, dimensional data, and sample G-code. This tool provides an efficient, user-friendly solution for creating 3D relief designs, making advanced modeling accessible to a broader user base including artists, students, and designers.

INDEX

CHAPTER-1

INTRODUCTION

The **3D Art Relief Design Application** is an advanced digital tool developed to facilitate the creation of **three-dimensional relief designs** from **2D images or digital artwork**. Designed for professionals in fields such as **digital sculpting, architectural ornamentation, CNC machining, and fine arts**, this application provides a seamless workflow for generating high-precision relief models with customizable depth, texture, and detailing.

CHAPTER-2

PROBLEM STATEMENT

Creating **3D relief designs** from **2D images** is a complex and time-consuming process requiring advanced modeling skills and specialized software. Traditional methods lack automation, making it difficult for artists, designers, and manufacturers to achieve **high precision and efficiency**. Existing tools often do not provide **real-time previews, seamless integration with digital fabrication, or user-friendly interfaces**. This results in prolonged design cycles, increased costs, and a steep learning curve. There is a need for a **simplified, automated, and efficient solution** that enables users to generate **high-quality 3D reliefs** with ease.

CHAPTER-3

SYSTEM REQUIREMENTS

WINDOWS:

Requirement	Minimum	Recommended
CPU cores	2	4
Ram in GB	4	8
Python Version	3.8+	3.8+
Free Disk space in MB	500	500
Display Resolution	1280*720	1920*1080

macOS:

Requirement	Minimum	Recommended
CPU cores	2	4
Ram in GB	4	8
Python Version	3.8	3.8+
Free Disk space in MB	500	500
Display Resolution	1280*720	Retina/4k

Linux:

Requirement	Minimum	Recommended
CPU cores	2	4
Ram in GB	4	8
Python Version	3.8	3.8+
Free Disk space in MB	500	500
Display Resolution	1280*720	1920*1080

CHAPTER-4
SOFTWARE SPECIFICATIONS

WINDOWS:

1. OS:

   Windows 10/11 (64-bit)

2. Software & Tools:

   Python 3.8.3

   Visual Studio Code

3. Libraries

   Streamlit, matplotlib, OpenCV, PIL (Pillow), Matplotlib, Plotly, Trimesh, STL (numpystl), ReportLab

MAC and Linux:

1. 1. OS:

   Windows 10/11 (64-bit)

2. Software & Tools:

   Python 3.8.3

   Visual Studio Code

3. Libraries

   Streamlit, matplotlib, OpenCV, PIL (Pillow), Matplotlib, Plotly, Trimesh, STL (numpy-stl), ReportLab

CHAPTER-5

FRAMEWORK

The Art Relief Design App is developed using the Streamlit framework, which is a Python based open-source tool designed for building interactive web applications. Streamlit simplifies the development process by allowing developers to create responsive user interfaces without the need for HTML, CSS, or JavaScript.

Streamlit provides a smooth and dynamic user experience, making it ideal for applications that involve real-time interaction, such as image uploading, height map visualization, 3D rendering, and file generation. Its clean layout and tab-based navigation make the Art Relief Design App accessible even to non-technical users.

Streamlit allows developers to build powerful web applications without the need for HTML, CSS, or JavaScript.

Streamlit supports live interaction and instant updates, making it perfect for apps with dynamic inputs and visualizations.

It works seamlessly with Python libraries like NumPy, OpenCV, Plotly, Matplotlib, and more ideal for image processing and 3D modeling.

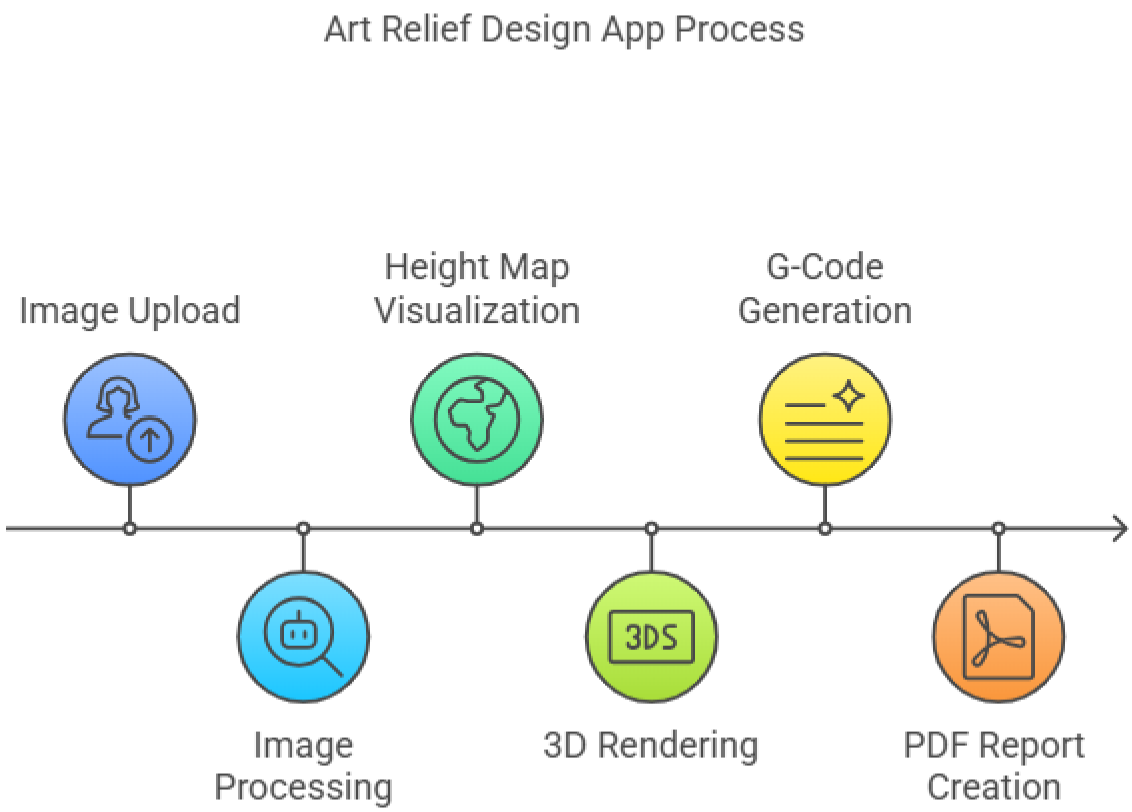
Users can access the app directly in their web browser, making it easy to use without installing any extra software.

Supporting Libraries and Tools:

- NumPy – Handles mathematical and array operations.
- OpenCV (cv2) – Used for image processing and grayscale conversion.
- PIL (Pillow) – Supports image loading and manipulation.
- Matplotlib & Plotly – Used for height map visualization and 3D previews.
- Trimesh & numpy-stl – For creating and exporting STL mesh files.
- ReportLab – Generates detailed PDF reports including images, statistics, and G-code preview.

CHAPTER-6:

WORKING MECHANISM



CHAPTER 7
METHODOLOGY

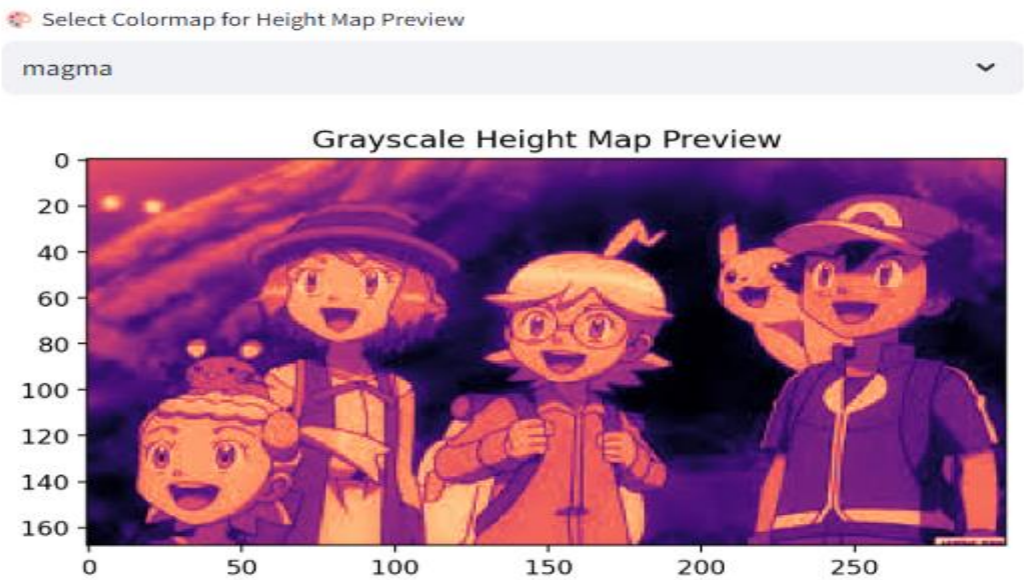
Tab 1: Image Upload & Selection

Purpose:

This tab allows users to upload 2D images that will be converted into 3D relief models.

Features:

1. Upload Image: Users can select and upload an image in formats like JPEG, PNG, or BMP.
2. Batch Processing Support: Users can upload multiple images at once for bulk conversion.
3. Preview Image: A thumbnail preview is displayed to verify the uploaded file.
4. Basic Adjustments: Users can crop, resize, or rotate the image before processing.
5. Proceed to Next Step: Once the image is selected, the user clicks "Next" to move to the processing phase.



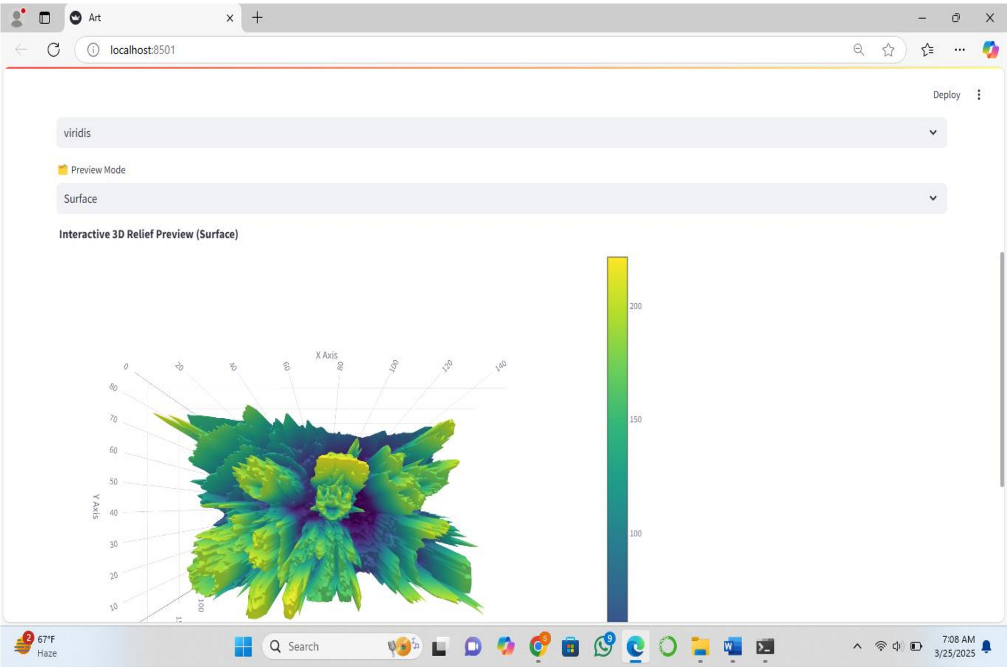
Tab 2: 3D Model Creation & Editing

Purpose:

This tab converts the height map into a **3D relief model (STL format)** and allows **basic modifications**.

Features & Workflow:

1. **Generate 3D Mesh:** The system converts the height map into a **3D surface model** using NumPy-STL and Trimesh.
2. **Depth & Texture Adjustments:** Users can fine-tune the **height, smoothness, and detailing** of the model.
3. **Material Simulation:** A **preview mode** shows the model with different material effects like **wood, metal, stone, and plastic**.
4. **Lighting Effects:** Users can adjust **shadows, highlights, and reflections** for better visualization.
5. **Rotate & Zoom Controls:** Interactive 3D view to inspect the model from all angles.



Tab 3: Export & File Generation

Purpose:

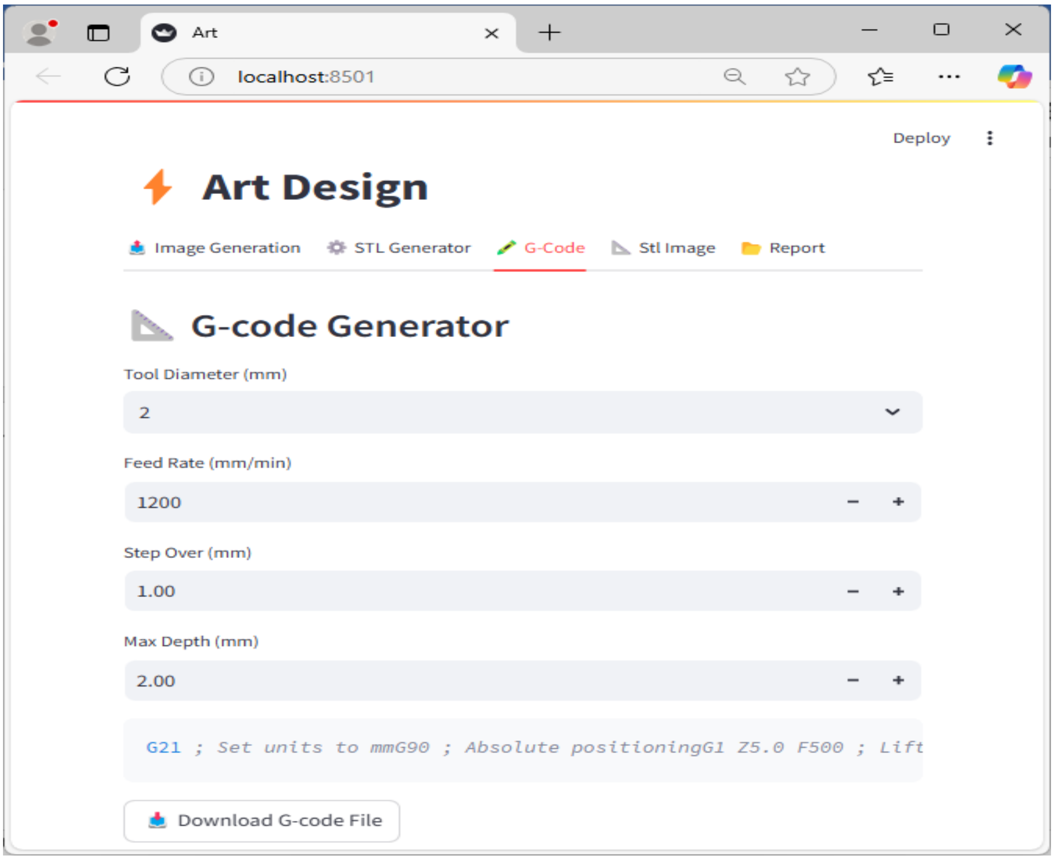
This tab allows users to export the final 3D relief model in different formats.

Features & Workflow:

1. **Export Options:**
    - STL File: For 3D printing or further editing in CAD software.
    - G-Code: For CNC machining and engraving.
    - PNG/JPG: For 2D visualization and sharing.

2. **Download:**

Users can download generated gcode (geometric code).



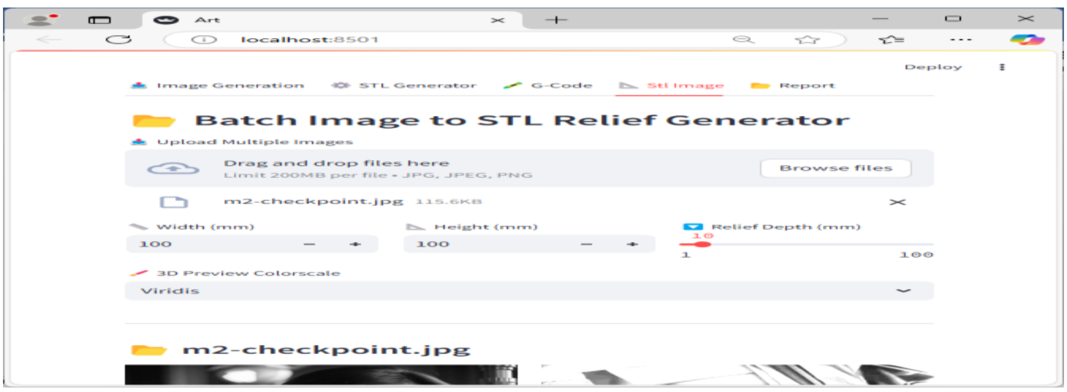
Tab 4: Batch Processing & Automation

Purpose:

This tab allows users to **process multiple images at once** for high-efficiency workflows.

Features & Workflow:

1. **Upload Multiple Images:** Users can select **multiple 2D images** for bulk processing.
2. **Automated Processing:** The application **automatically applies preprocessing, height map generation, and 3D model conversion** to all images.
3. **Batch Preview:** Users can view all generated models in a **grid layout**.
4. **Bulk Export:**
   - Users can **download all STL files at once**.
   - A **single PDF report** is generated for all processed models.
5. **Time & Efficiency Optimization:** This feature is ideal for **designers, manufacturers, and CNC operators** who need **quick and large-scale 3D model creation**.



CHAPTER-8
SOURCE CODE

Python:

The backend for this webpage is made with python and stored as art3.py file and uploaded in the GitHub to get the link.

PYTHON CODE:

```
import streamlit as st
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import io
import os
import tempfile
import cv2
import trimesh
import plotly.graph_objects as go
from stl import mesh as stl_mesh

st.set_page_config(layout="wide", page_title="Art")

st.title("⚡ Art Design")

# Tabs
tab1, tab2, tab3, tab4, tab5 = st.tabs([
    "🧍 Image Generation",
    "⚙ STL Generator",
    "🖊 G-Code",
    "📐 Stl Image",
    "📁 Report"
])

# ------------------ TAB 1 - Upload & Relief Generator ------------------
with tab1:
    st.header("🖼 Height map")
    uploaded_image = st.file_uploader("Upload an image", type=["jpg", "jpeg", "png", "bmp", "tif", "tiff"])

    if uploaded_image:
```

```python
        image = Image.open(uploaded_image)
        st.session_state['original_image'] = image
        st.image(image, caption="Original Image", use_column_width=True)

        image_array = np.array(image)
        if len(image_array.shape) == 3:
            gray_image = cv2.cvtColor(image_array, cv2.COLOR_RGB2GRAY)
        else:
            gray_image = image_array

        gray_image = gray_image.astype(float)

        colormap_options = ["gray", "viridis", "plasma", "inferno", "magma", "cividis"]
        selected_colormap = st.selectbox("🎨 Select Colormap for Height Map Preview", colormap_options, index=0)

        fig, ax = plt.subplots()
        im = ax.imshow(gray_image, cmap=selected_colormap)
        ax.set_title("Grayscale Height Map Preview")
        st.pyplot(fig)

        st.session_state['relief'] = gray_image


# ------------------ TAB 2 - STL Generator --
with tab2:
    st.header("🧊 STL Generator")

    if 'relief' in st.session_state:
        relief = st.session_state['relief']
        rows, cols = relief.shape
        st.write(f"Relief Shape: {rows} x {cols}")

base_thickness = st.number_input("Base Thickness (mm)", min_value=0.0, max_value=10.0, value=2.0, step=0.5)
add_base_plate = st.checkbox("Add Base Plate", value=True)
scale_percent = st.slider("Preview Resolution Scale (%)", 10, 100, 50, 10)

# Downscale for preview
scaled_relief = cv2.resize(relief, (0, 0), fx=scale_percent/100, fy=scale_percent/100, interpolation=cv2.INTER_AREA)

# Colormap and Preview Mode Selection
colormap_options = ["gray", "viridis", "plasma", "inferno", "magma", "cividis"]
selected_colormap = st.selectbox("🎨 Select Colormap for 3D STL Preview", colormap_options, index=1)
preview_mode = st.selectbox("🗂️ Preview Mode", ["Surface", "Wireframe", "Heightmap"], index=0)
```

```python
def generate_stl_fast(relief_data, base_thickness=2.0, add_base=True):
    relief_data = np.flipud(relief_data)
    rows, cols = relief_data.shape
    x = np.linspace(0, cols, cols)
    y = np.linspace(0, rows, rows)
    X, Y = np.meshgrid(x, y)
    Z = relief_data.astype(float)
    if add_base:
        Z += base_thickness
    vertices = np.column_stack((X.flatten(), Y.flatten(), Z.flatten()))
    faces = []
    for i in range(rows - 1):
        for j in range(cols - 1):
            idx = i * cols + j
            faces.append([idx, idx + 1, idx + cols])
            faces.append([idx + 1, idx + cols + 1, idx + cols])
    faces = np.array(faces)
    mesh_obj = trimesh.Trimesh(vertices=vertices, faces=faces)
    return mesh_obj, X, Y, Z

# Generate preview mesh
preview_mesh, X, Y, Z = generate_stl_fast(scaled_relief, base_thickness, add_base=add_base_plate)

# Interactive 3D Plotly Preview
if preview_mode == "Surface":
    plotly_fig = go.Figure(data=[go.Surface(z=Z, x=X, y=Y, colorscale=selected_colormap)])
elif preview_mode == "Wireframe":
    plotly_fig = go.Figure(data=[go.Surface(
        z=Z, x=X, y=Y, colorscale=selected_colormap, showscale=False,
        contours={"z": {"show": True, "start": np.min(Z), "end": np.max(Z), "size": 1}}
elif preview_mode == "Heightmap":
    plotly_fig = go.Figure(data=[go.Contour(z=Z, x=X[0], y=Y[:,0], colorscale=selected_colormap)])

plotly_fig.update_layout(
    title=f"Interactive 3D Relief Preview ({preview_mode})",
    scene=dict(
        xaxis_title='X Axis',
        yaxis_title='Y Axis',
        zaxis_title='Z Axis'
    ),
    width=900,
    height=700,
    margin=dict(l=10, r=10, b=10, t=30)
)
st.plotly_chart(plotly_fig)
```

```python
    # Export full-resolution STL file
    full_mesh, _, _, _ = generate_stl_fast(relief, base_thickness, add_base=add_base_plate)
    stl_path = tempfile.NamedTemporaryFile(delete=False, suffix=".stl").name
    full_mesh.export(stl_path)

    with open(stl_path, "rb") as f:
        st.download_button("📥 Download High-Res STL", f, file_name="relief_output.stl")

else:
    st.warning("⚠️ Please generate a relief in Tab 1 first.")
```

```python
# ------------------ TAB 3 - G-code Generator ------------------
with tab3:
    st.header("📐 G-code Generator")
    if 'relief' in st.session_state:
        tool_diameter = st.selectbox("Tool Diameter (mm)", [2, 4, 6, 8])
        feed_rate = st.number_input("Feed Rate (mm/min)", 100, 5000, 1200, 100)
        step_over = st.number_input("Step Over (mm)", 0.1, float(tool_diameter), 1.0, 0.1)
        max_depth = st.number_input("Max Depth (mm)", 0.1, 10.0, 2.0, 0.1)

        relief = st.session_state['relief']
        gcode = ["G21 ; Set units to mm", "G90 ; Absolute positioning", "G1 Z5.0 F500 ; Lift tool"]

        step_px = max(1, int(step_over))
        for i in range(0, relief.shape[0], step_px):
            row = relief[i]
            for j in range(0, relief.shape[1]):
                x = j
```

```python
            for j in range(0, relief.shape[1]):
                x = j
                y = i
                z = -min(row[j], max_depth)
                gcode.append(f"G1 X{x:.2f} Y{y:.2f} Z{z:.2f} F{feed_rate}")

        gcode.append("G1 Z5.0 ; Lift tool at end")
        gcode.append("M30 ; End of program")
        gcode_output = "".join(gcode)
        st.code(gcode_output, language='gcode')
        st.download_button("📥 Download G-code File", gcode_output.encode(), file_name="relief_output.nc")
    else:
        st.warning("⚠️ Generate relief first in Tab 1.")
```

```python
with tab4:
    st.header("📁 Batch Image to STL Relief Generator")

    uploaded_files = st.file_uploader("📤 Upload Multiple Images", type=["jpg", "jpeg", "png"], accept_multiple_files=True)

    col1, col2, col3 = st.columns(3)
    with col1:
        width = st.number_input("📏 Width (mm)", min_value=10, max_value=1000, value=100)
    with col2:
        height = st.number_input("📐 Height (mm)", min_value=10, max_value=1000, value=100)
    with col3:
        depth = st.slider("🔽 Relief Depth (mm)", min_value=1, max_value=100, value=10)

    colorscale_options = {
        "Viridis": "Viridis",
        "Cividis": "Cividis",
        "Inferno": "Inferno",
        "Plasma": "Plasma",
        "Magma": "Magma",
        "Greys": "Greys",
        "YlGnBu": "YlGnBu",
        "YlOrRd": "YlOrRd"
    }
    selected_colorscale = st.selectbox("🖌️ 3D Preview Colorscale", list(colorscale_options.keys()), key="batch_colormap")

    def render_3d_preview(X, Y, Z, colormap):
        fig = go.Figure(data=[go.Surface(z=Z, x=X, y=Y, colorscale=colormap)])
        fig.update_layout(
            scene=dict(zaxis_title='Depth', xaxis_title='X', yaxis_title='Y'),
            margin=dict(l=0, r=0, b=0, t=0), height=600
        )
```

```python
if uploaded_files:
    for uploaded_file in uploaded_files:
        st.divider()
        st.subheader(f"📁 {uploaded_file.name}")
        image = Image.open(uploaded_file).convert("L")
        img_array = np.array(image)
        img_array = np.flipud(img_array)

        height_map = (img_array - np.min(img_array)) / (np.max(img_array) - np.min(img_array)) * depth
        X = np.linspace(0, width, img_array.shape[1])
        Y = np.linspace(0, height, img_array.shape[0])
        X, Y = np.meshgrid(X, Y)
        Z = height_map

        col_img, col_map = st.columns(2)
        with col_img:
            st.image(image, caption="Grayscale Input Image", use_column_width=True)
        with col_map:
            st.image(height_map, caption="Heightmap Visualization", use_column_width=True, clamp=True)

        fig = render_3d_preview(X, Y, Z, colorscale_options[selected_colorscale])
        st.plotly_chart(fig, use_container_width=True)

        stl_data = []
        for i in range(Z.shape[0] - 1):
            for j in range(Z.shape[1] - 1):
                p1 = [X[i][j], Y[i][j], Z[i][j]]
                p2 = [X[i][j + 1], Y[i][j + 1], Z[i][j + 1]]
                p3 = [X[i + 1][j], Y[i + 1][j], Z[i + 1][j]]
                p4 = [X[i + 1][j + 1], Y[i + 1][j + 1], Z[i + 1][j + 1]]
                stl_data.append([p1, p2, p3])
```

```python
            stl_array = np.zeros(len(stl_data), dtype=stl_mesh.Mesh.dtype)
            for i, f in enumerate(stl_data):
                stl_array["vectors"][i] = np.array(f)

            model_mesh = stl_mesh.Mesh(stl_array)
            stl_filename = f"{os.path.splitext(uploaded_file.name)[0]}.stl"
            stl_path = os.path.join(tempfile.gettempdir(), stl_filename)
            model_mesh.save(stl_path)

            with open(stl_path, "rb") as f:
                st.download_button(f"💾 Download STL for {uploaded_file.name}", f, file_name=stl_filename)

            os.remove(stl_path)


from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from reportlab.lib.utils import ImageReader


with tab5:
    st.header("📄 Relief Report")
    if 'relief' in st.session_state:
        relief = st.session_state['relief']

        st.subheader("📷 Original Image")
        if 'original_image' in st.session_state:
            st.image(st.session_state['original_image'], caption="Original Uploaded Image")
            original_img_path = os.path.join(tempfile.gettempdir(), "original_image_temp.png")
            st.session_state['original_image'].save(original_img_path)


        st.subheader("🗺️ Grayscale Height Map")
        fig, ax = plt.subplots()
        ax.imshow(relief, cmap='gray')
        ax.axis('off')
        ax.set_title("Relief Height Map")
        heightmap_path = os.path.join(tempfile.gettempdir(), "heightmap_temp.png")
        fig.savefig(heightmap_path, bbox_inches='tight', dpi=150)
        st.pyplot(fig)

        st.subheader("📐 Relief Dimensions")
        min_h = np.min(relief)
        max_h = np.max(relief)
        avg_h = np.mean(relief)
        stats_text = f"""
Relief Dimensions:
Rows x Columns: {relief.shape[0]} x {relief.shape[1]}
Min Height: {min_h:.2f} mm
Max Height: {max_h:.2f} mm
Average Height: {avg_h:.2f} mm
"""
        st.text(stats_text)

        st.subheader("📊 Relief Height Histogram")
        fig2, ax2 = plt.subplots()
        ax2.hist(relief.flatten(), bins=50, color='skyblue', edgecolor='black')
        ax2.set_title("Height Distribution")
        histogram_path = os.path.join(tempfile.gettempdir(), "histogram_temp.png")
        fig2.savefig(histogram_path, bbox_inches='tight', dpi=150)
        st.pyplot(fig2)
```

```
gcode_text = st.session_state.get('gcode', None)
if gcode_text:
    st.subheader("🖨 G-code Preview")
    st.code(gcode_text, language='gcode')
    gcode_lines = gcode_text.strip().split('\n')
else:
    gcode_lines = []

# === PDF GENERATION ===
pdf_path = os.path.join(tempfile.gettempdir(), "relief_report.pdf")
c = canvas.Canvas(pdf_path, pagesize=A4)
pdf_width, pdf_height = A4
margin = 40
y = pdf_height - margin

# Title
c.setFont("Helvetica-Bold", 18)
c.drawString(margin, y, "Relief Report")
y -= 30

# Stats block
c.setFont("Helvetica", 12)
for line in stats_text.strip().split("\n"):
    c.drawString(margin, y, line.strip())
    y -= 18

# Add images compactly
if os.path.exists(original_img_path):
    y -= 10
    c.drawImage(ImageReader(original_img_path), margin, y - 120, width=180, height=120, mask='auto')

      if os.path.exists(heightmap_path):
          c.drawImage(ImageReader(heightmap_path), margin + 200, y - 120, width=180, height=120, mask='auto')
          c.drawString(margin + 200, y - 130, "Height Map")

      y -= 160

      if os.path.exists(histogram_path):
          c.drawImage(ImageReader(histogram_path), margin, y - 150, width=360, height=150, mask='auto')
          c.drawString(margin, y - 160, "Relief Height Histogram")
          y -= 180

      # Add G-code Preview (up to 40 lines max)
      if gcode_lines:
          c.setFont("Courier", 8)
          c.drawString(margin, y, "G-code Preview (First 40 lines):")
          y -= 14
          for line in gcode_lines[:40]:
              if y < 50:
                  break
              c.drawString(margin, y, line)
              y -= 10

      c.save()

      with open(pdf_path, "rb") as f:
          st.download_button("📥 Download Relief Report (PDF)", f, file_name="relief_report.pdf", mime="application/pdf")

      st.success("✅ Report generated as a single-page PDF.")
  else:
      st.warning("⚠️ Please generate relief from Tab 1 first.")
```

CHAPTER-9

OUTPUT

Webpage:

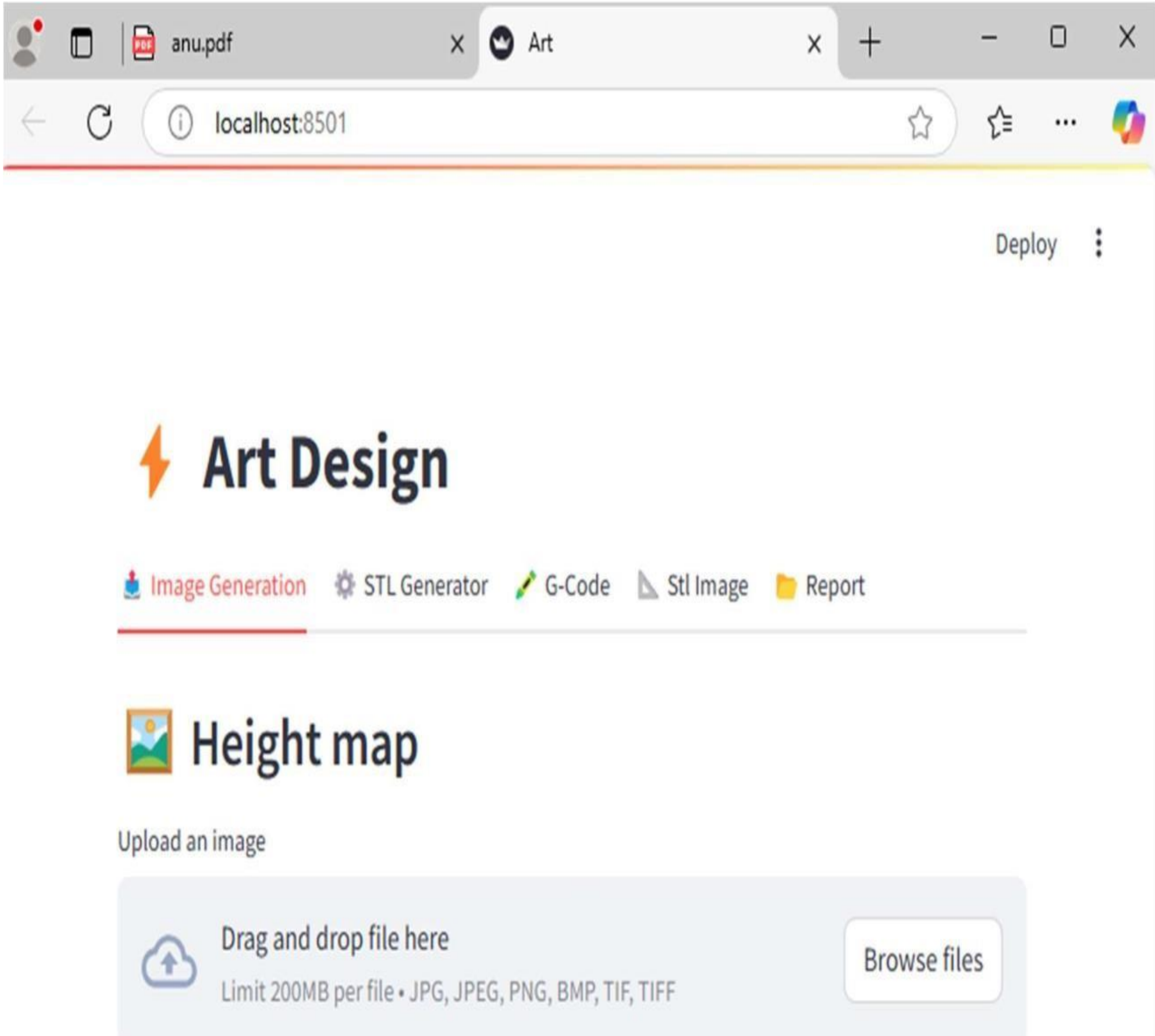
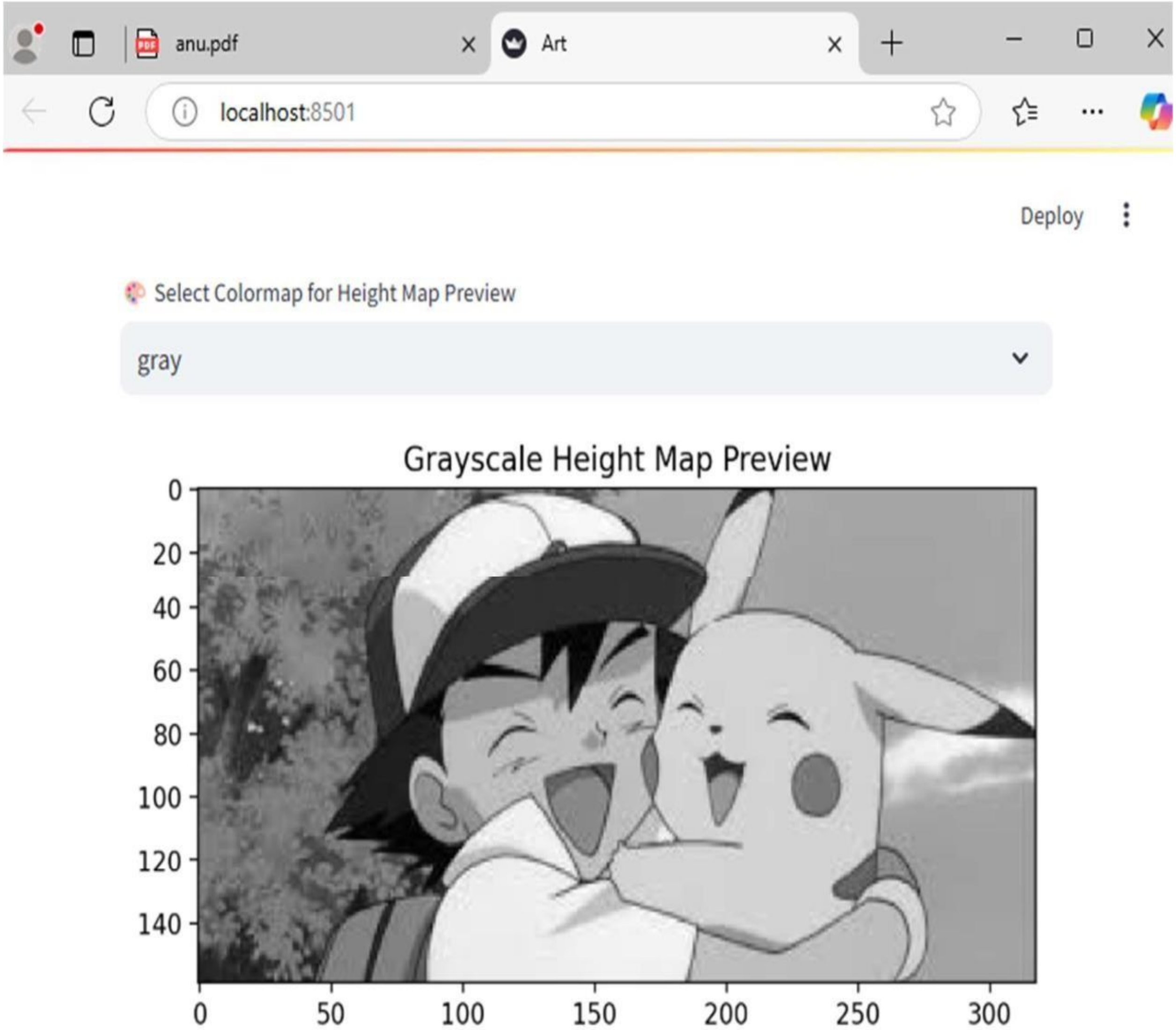
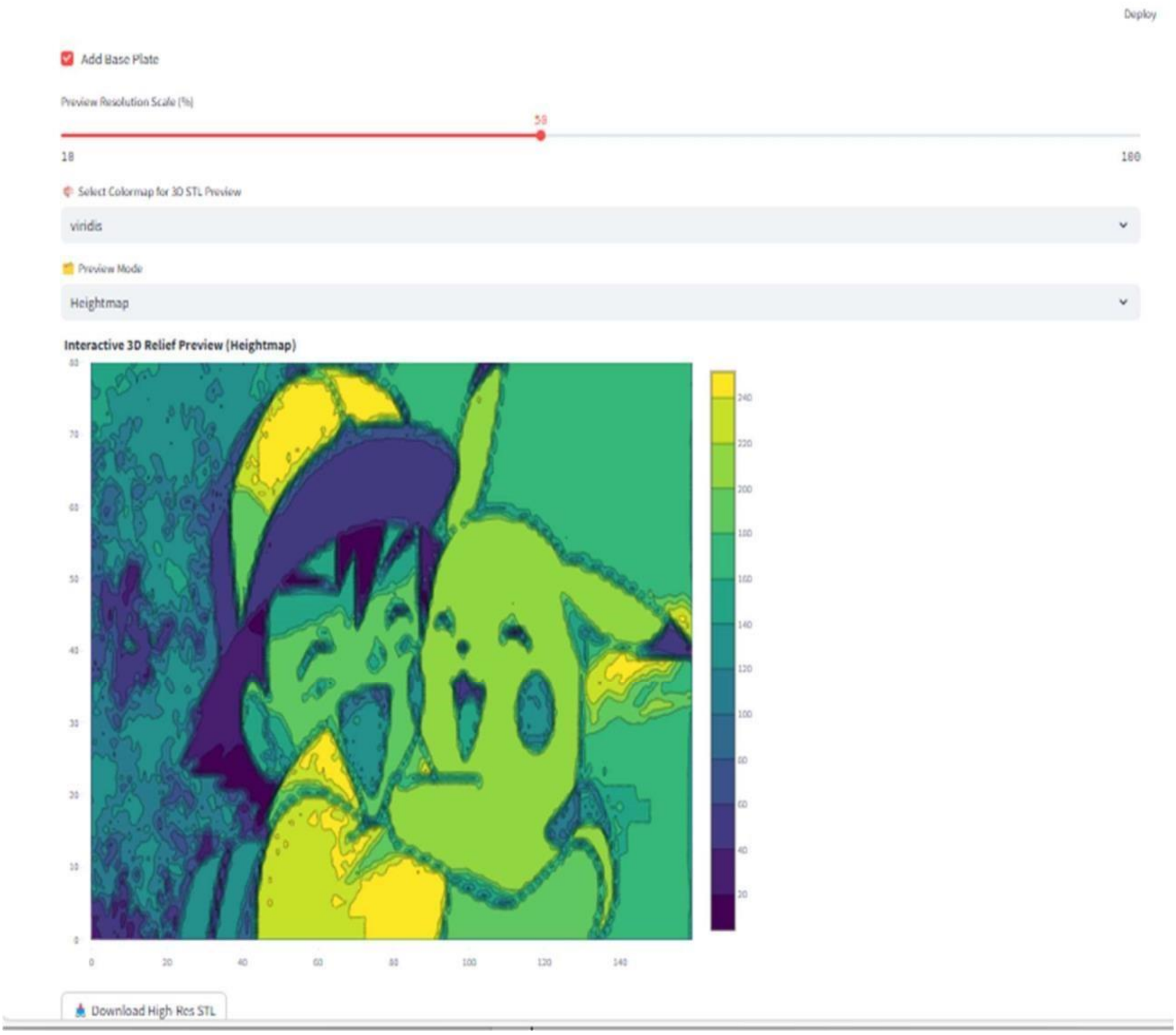


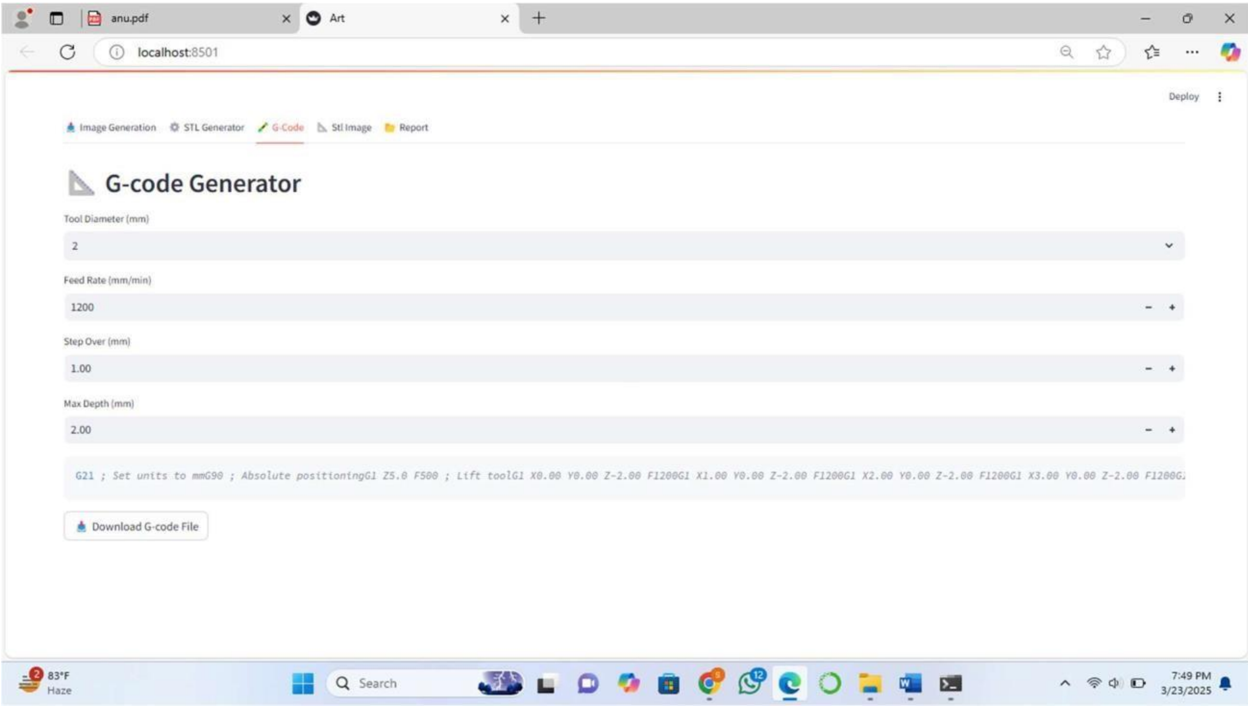
Image Generation:



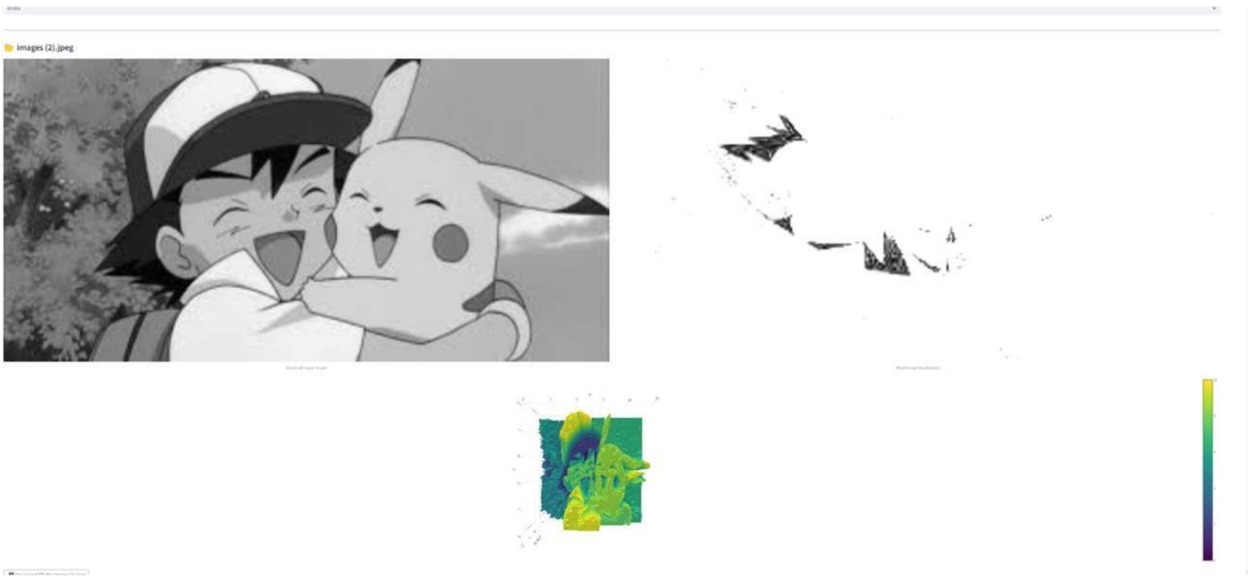
Stl Generation:



G-Code:



Batch Generation:



Report:

Relief Report

Relief Dimensions:

Rows x Columns: 159 x 318

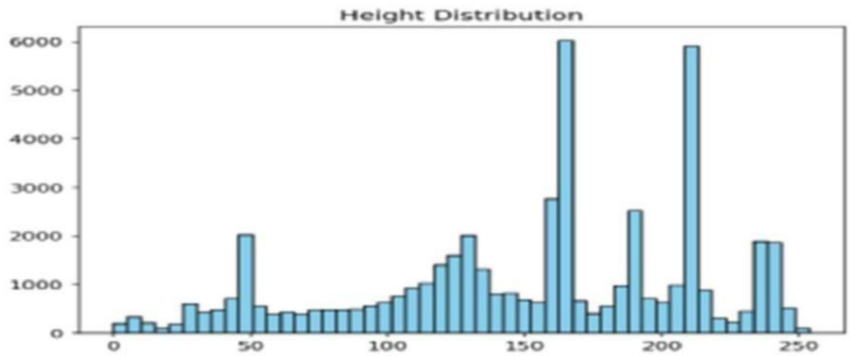
Min Height: 0.00 mm

Max Height: 254.00 mm

Average Height: 151.42 mm

Original Image

Height Map



Relief Height Histogram

CHAPTER-10

CONCLUSION

The **3D Art Relief Design Application** provided an easy and efficient way to convert **2D images into 3D relief models**. It uses **image processing, AI-based depth mapping, and texture application** to create detailed and accurate 3D designs. The system is suitable for both **professionals and beginners**, making the design process **faster and more efficient**. With its automated steps and simple interface, the application helps improve **accuracy, creativity, and productivity** in 3D modeling.

CH-11: REFERENCES

1. Streamlit Documentation – https://docs.streamlit.io
2. NumPy Official Site – https://numpy.org
3. OpenCV Official Site – https://opencv.org
4. Pillow (PIL) Documentation –https://pillow.readthedocs.io
5. 5. Matplotlib Documentation – https://matplotlib.org
6. Trimesh Library – https://trimsh.org
7. NumPy-STL Package – https://pypi.org/project/numpy-stl
8. ReportLab Toolkit for PDF – https://www.reportlab.com
9. Python Official Website – https://www.python.org