CS-3107 PROJECT REPORT

IMAGE QUILTING FOR TEXTURE SYNTHESIS

Group - 13

INTRODUCTION

Texture synthesis is the process of generating a larger image or texture based on a given smaller example, such that the new texture maintains the visual and structural properties of the original. The goal is to create a seamless and plausible expansion of the original sample.

Image quilting is a specific algorithm or method used for texture synthesis. It works by stitching together small patches of texture from the input image to create a larger output texture. The patches are carefully aligned to ensure smooth transitions between adjacent patches, minimizing visible seams.

OVERVIEW

This code implements a texture synthesis technique using patch-based quilting based on the paper "**Image Quilting for Texture Synthesis and Transfer**" by **Efros & Freeman**.

It allows the generation of larger textures or filling regions using small input texture patches. The synthesis can use random patch placement, error minimization (overlap boundaries), or optimal patch blending via min-cut paths.

KEY FUNCTIONS OF THE CODE

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import math
import heapq
```

1. `numpy (np)`: Efficiently handles numerical operations, such as matrix manipulation, crucial for patch extraction and error computation.

2. **`cv2`**: Provides tools for reading, displaying, and processing images used in patch selection and visualization.
3. **`google.colab.patches.cv2_imshow`**: Displays intermediate results of the quilting process in Google Colab.
4. **`math`**: Supplies mathematical functions like `sqrt`, aiding calculations in overlap blending.
5. **`heapq`**: Implements a priority queue for efficiently finding minimal error paths in seam-cutting.

STEP 1: Random Plcaement

```
def randomPatch(texture, patchLength):
```

- **Purpose:** Select a random patch of a given size from the texture.
- **Parameters:**
  - **`texture`**: Input texture as a 3D NumPy array.
  - **`patchLength`**: Side length of the square patch to extract.
- **Returns:** A random square patch from the texture.

STEP 2: Overlap Boundaries

```
def L2OverlapDiff(patch, patchLength, overlap, res, y, x):
```

- **Purpose**: Compute the L2 (squared) overlap error for a patch based on its overlapping regions with the existing quilted texture.
- **Parameters:**
  - **`patch`**: Candidate patch for placement.
  - **`patchLength`**: Side length of the square patch to extract.
  - **`overlap`**: Overlap width in pixels.
  - **`res`**: Current quilted texture
  - **`y, x`**: Position in red where the patch would be placed.
- **Returns:** Overlap error as a scalar value.

```
def randomBestPatch(texture, patchLength, overlap, res, y, x):
```

- **Purpose:** Select a patch with minimal overlap error from the texture.
- **Parameters:** Same as **`L2OverlapDiff`**.
- **Returns:** Best-matching patch with minimal overlap error.

STEP 3: Minimum Error Cut

```
def minCutPath(errors):
```

- **Purpose:** Find the vertical path through a 2D error matrix with the minimum cumulative error.
- **Algorithm:** Implements Dijkstra's shortest path algorithm.
- **Parameters:**
    - `errors`: 2D matrix of overlap errors.
- **Returns:** List of indices representing the vertical min-cut path.

```
def minCutPatch(patch, patchLength, overlap, res, y, x):
```

- **Purpose:** Blend a patch into the existing texture using min-cut paths for smoother transitions.
- **Parameters:** Same as `L2OverlapDiff`.
- **Returns:** Modified patch with blended boundaries.

CODE EXPLANATION

```
texture = cv2.imread("/content/input_73.png")
cv2_imshow(texture)
```

The image file loads into the texture variable as a NumPy array

```
cv2_imshow(quilt(texture, 64, (8, 8), "random"))
```

A texture quilt is created by combining `64-pixel` patches in an `8x8` grid. Image Quilting is done in `random` mode.

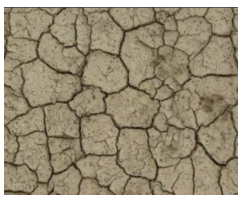
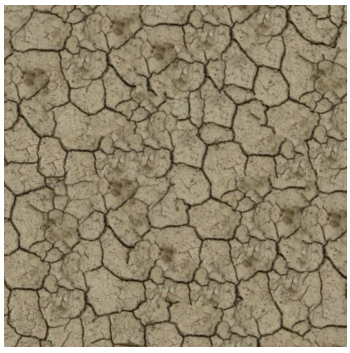
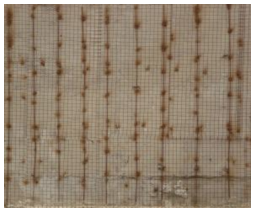
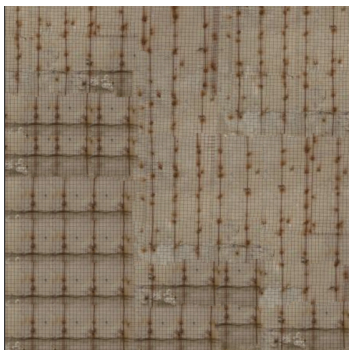
```
cv2_imshow(quilt(texture, 64, (8, 8), "best"))
```

The texture is processed using `best` mode, where patches are selected to minimize overlap error with previously placed patches.

```
cv2_imshow(quilt(texture, 64, (8, 8), "cut"))
```

The texture is processed using `cut mode`, where the patches are blended using minimum-error paths for smooth transitions.

SOME RESULTS

EVALUATION OF DIFFERENT ALGORITHMS

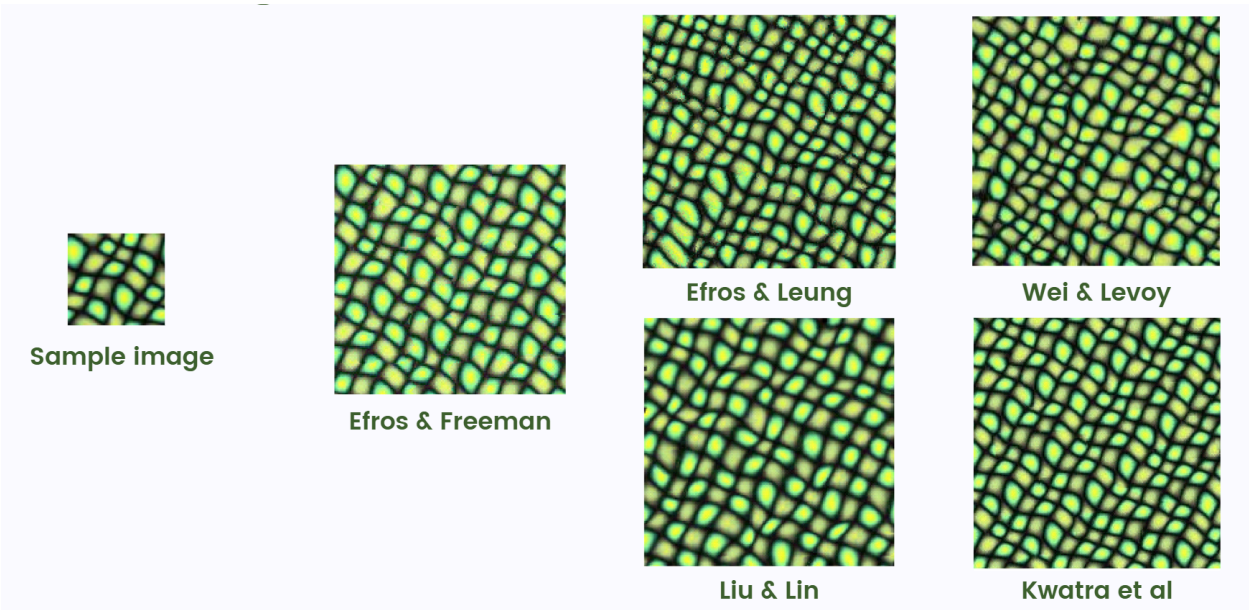
Evaluation Metrics

Overlap Error

- **Description**: Quantify the error in overlapping regions between patches.
- **Metric**: Compute the **mean squared error (MSE)** or **sum of squared differences (SSD)** in the overlap areas.

Evaluation Table

Method	**Key Idea**	**Speed**	**Seam Quality**	**Complex Texture Handling**
Efros & Leung	Pixel-by-pixel synthesis	Slow	Moderate	Good
Efros & Freeman	Patch-based quilting with boundary cuts	Moderate	Good	Moderate
Wei & Levoy	Random patch-based synthesis	Fast	Moderate	Moderate
Kwatra et al.	Patch synthesis with graph cuts	Slow	Excellent	Good
Liu & Lin	Pyramid-based optimization	Very slow	Excellent	Excellent



APPLICATIONS

1. Video Games and Virtual Reality

- **Terrain and Environment Design**:
  - Creating realistic landscapes (grass, rocks, water, or sand) by synthesizing textures from small samples.
- **Character Design**:
  - Synthesizing skin, fur, or fabric textures for game characters.

2. Film and Animation

- **CGI Textures**:
  - Generating realistic textures for props, clothing, or environmental elements in animated movies and films.

3. Architecture and Interior Design

- **Material Simulations**:
  - Extending scanned samples of wood grain, marble, or tiles to create realistic textures for virtual models.
- **Virtual Showrooms**:

    - Rendering high-quality images for interior decor elements like wallpapers, flooring, or fabric patterns.

4. Fashion and Textile Design

- **Pattern Generation**:
    - Extending or modifying small fabric swatches to visualize how designs will look on larger surfaces like garments.

5. Image Editing and Restoration

- **Inpainting**:
    - Fill in missing or damaged parts of images by synthesizing textures similar to the surrounding areas.
- **Content-Aware Fill**:
    - Tools like those in Adobe Photoshop use texture synthesis to remove unwanted objects and seamlessly fill the background.

6. Scientific Visualization

- **Medical Imaging**:
    - Synthesizing textures for anatomical structures (e.g., skin, organs) in 3D medical imaging for training or simulation.
- **Geology and Terrain Modeling**:
    - Simulating rock or soil textures for geoscience studies and environmental modelling.

7. Urban Planning and GIS

- **Satellite Image Filling**:
    - Filling missing regions in satellite imagery using texture synthesis.
- **Urban Simulations**:
    - Creating textures for urban models, such as buildings or streets, for visualization.

8. Product Design

- **Packaging and Branding**:
    - Synthesizing textures for product visualizations, especially when physical

prototypes are not yet available.

- **Surface Design**:
    - Extending materials like leather or plastic textures for 3D renderings of products.

10. Art and Creative Media

- **Digital Art**:
    - Artists use texture synthesis to create intricate patterns and backgrounds in digital artworks.
- **Generative Art**:
    - Using texture synthesis algorithms for procedurally generating artworks or immersive installations.

FUTURE SCOPE

The future scope of image quilting lies in advancing texture synthesis for realistic image generation, enabling applications in **gaming, virtual reality, and film production**. It also holds potential in **medical imaging and material science** for creating detailed simulations and enhancing visualization techniques.

REFERENCES

Efros & Freeman https://people.eecs.berkeley.edu/~efros/research/quilting/quilting.pdf

Liu & Lin https://www.cs.cmu.edu/~wclin/texture/texture03.pdf

TEAM DETAILS

SE22UCSE073	DACHEPALLY SATHWIK
SE22UCSE074	DAKOJI SESHA SAI PRANEETH
SE22UCSE075	DANDE MUKUND LAL
SE22UCSE076	DANTU MYTHRI SRI BHAVANI
SE22UCSE077	DEEPAK B