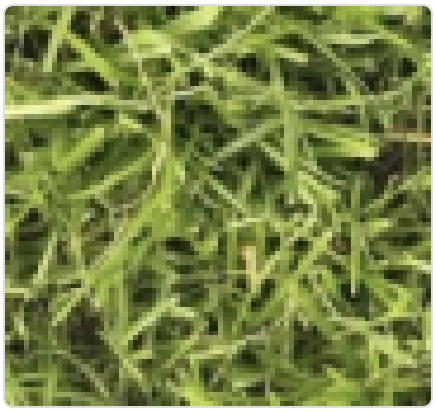
# CS 180 Image Quilting

## Project Overview

This project implements the image quilting algorithm for texture synthesis and transfer, as described in the SIGGRAPH 2001 paper by Efros and Freeman. The key idea is to generate textures by sampling overlapping patches from a given sample texture. This process is extended to texture transfer, where the texture of a sample is applied to a target image.

## Randomly Sampled Texture

The `quilt_random` function generates an output image by randomly sampling patches of size `patch_size` from a given sample image. Each patch is placed without any overlap, leading to visible seams. This approach serves as a baseline for more advanced methods.

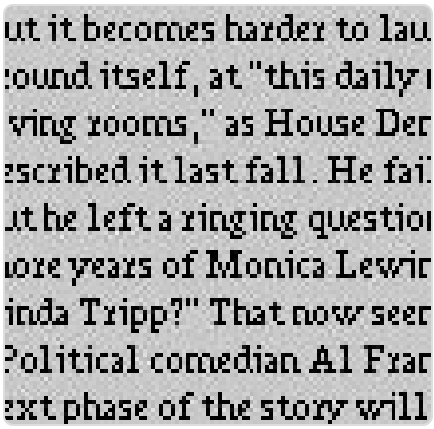


Grass

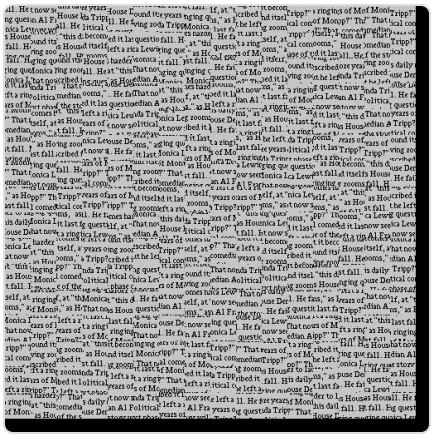
Grass Randomly Sampled

Brick

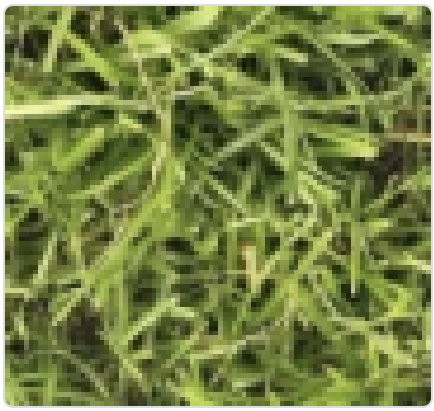
Brick Randomly Sampled



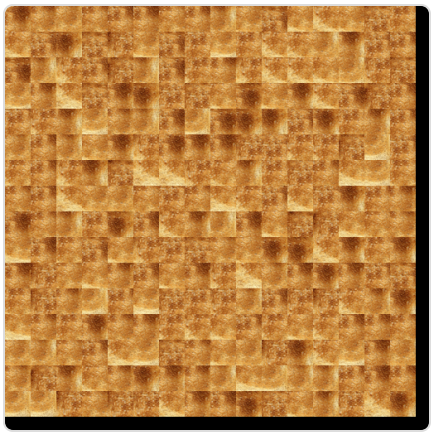
Text



Text Randomly Sampled



Toast

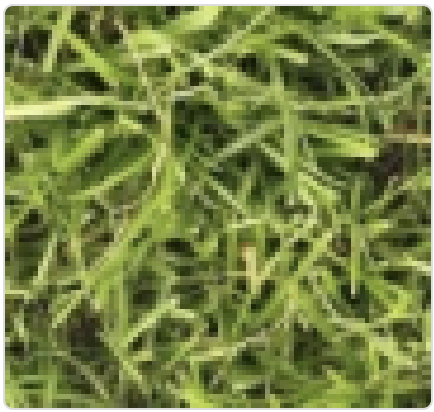


Toast Randomly Sampled

# Overlapping Patches

The `quilt_simple` function improves upon the random sampling method by overlapping patches.

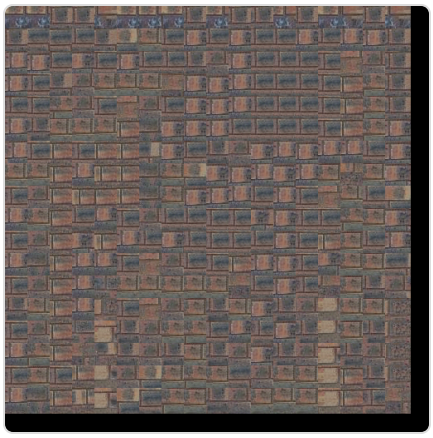
Using the `ssd_patch` and `choose_sample` helper functions, the algorithm ensures better alignment between adjacent patches, reducing visible seams.



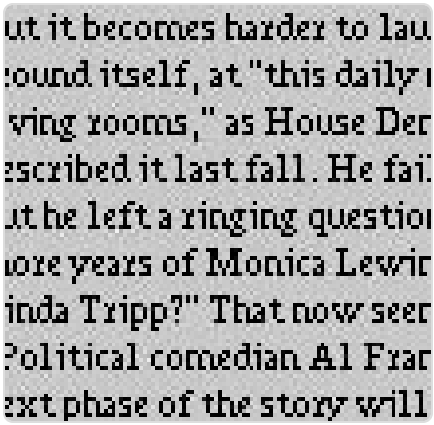
Grass

Grass, Overlapping Patches

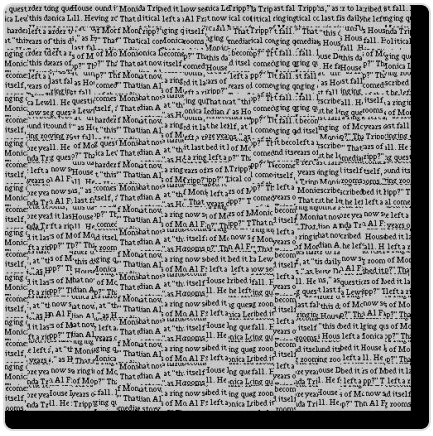
Brick



Brick, Overlapping Patches

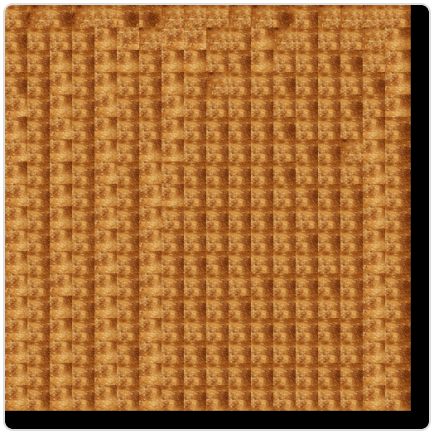



Text



Text, Overlapping Patches

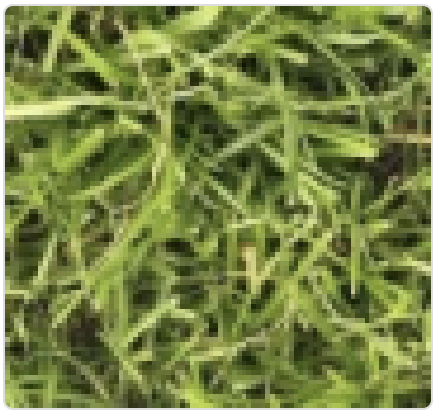
Toast



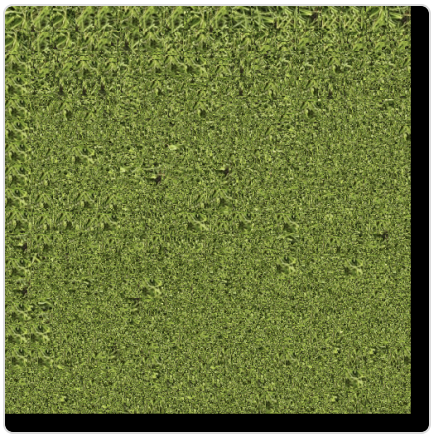
Toast, Overlapping Patches

# Seam Finding

The `quilt_cut` function further refines texture synthesis by incorporating seam finding. This minimizes edge artifacts by finding the minimum-cost path through overlapping regions using the `cut` function. The seams are blended to achieve smoother transitions.

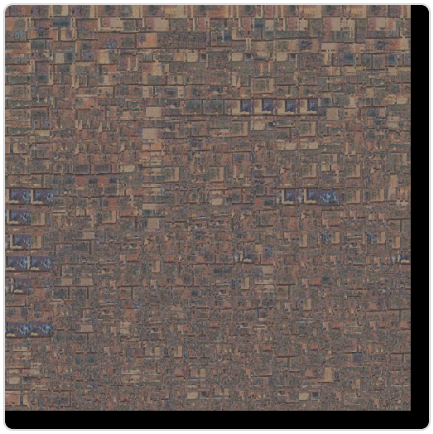


Grass

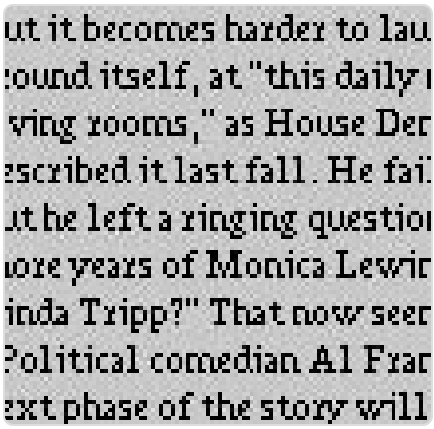


Grass, Seam Finding

Brick

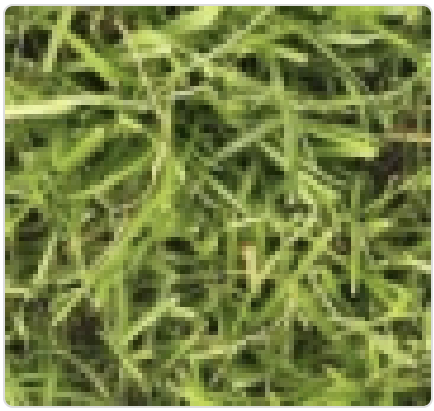


Brick, Seam Finding

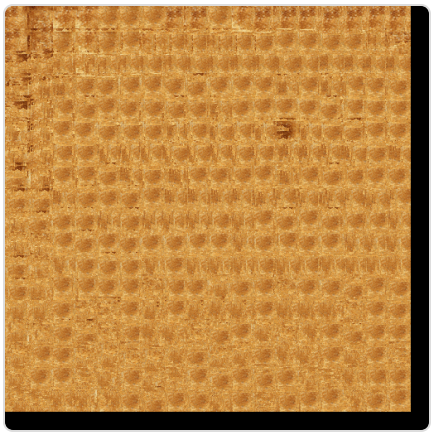



Text

Text, Seam Finding



Toast

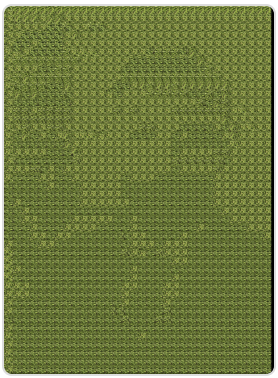


Toast, Seam Finding

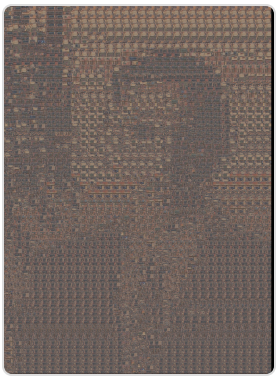
# Texture Transfer

The `texture_transfer` function applies texture synthesis to a target image. By balancing overlap costs

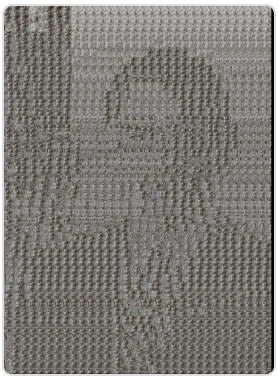
and target alignment costs, the algorithm overlays the texture while preserving the target's structural features.



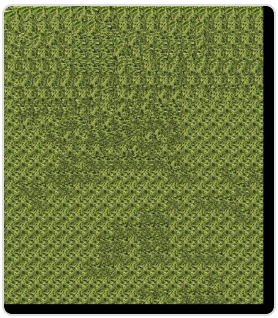
Grass Texture on Obama



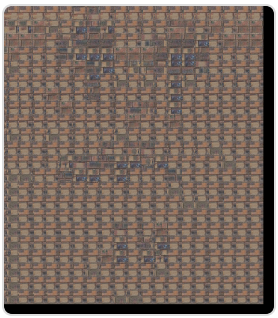
Brick Texture on Obama



White Texture on Obama



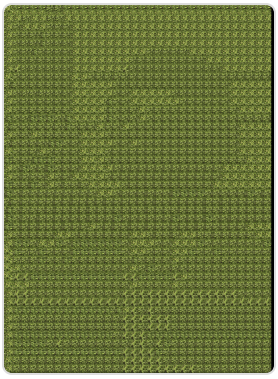
Grass Texture on Mickey Mouse



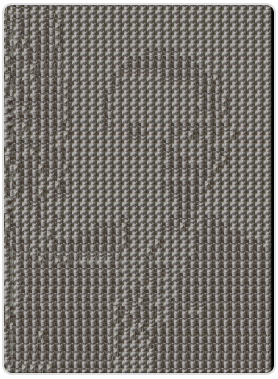
Brick Texture on Mickey Mouse

# Bells and Whistles: Iterative Texture Transfer

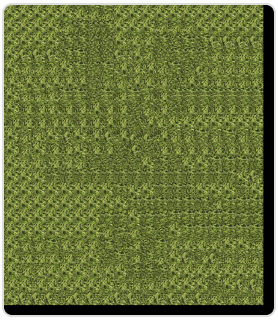
For the B&W portion of the project, I implemented the `texture_transfer_iterative` function. This method progressively refines the transfer result over multiple iterations by iteratively increasing the weight of the target image alignment term. This ensures a more accurate texture alignment with the target structure while retaining the visual characteristics of the texture sample.



Grass Texture on Obama, Iteratively



White Texture on Obama, Iteratively



Grass Texture on Mickey Mouse, Iteratively

# Improvements

Some of the challenges faced during the implementation included:

- Balancing the overlap cost ( `alpha` ) and target alignment cost.
- Implementing efficient seam finding to avoid misaligned patches, since I kept running into misalignment-related errors.

Future improvements could involve advanced blending techniques (e.g., Laplacian pyramids) and extending the algorithm to handle arbitrary shapes for image completion tasks.

# Acknowledgements

This project is a course project for CS 180 at UC Berkeley. Part of the starter code is provided by course staff. This website template is referenced from Bill Zheng.