15-418 Project Proposal

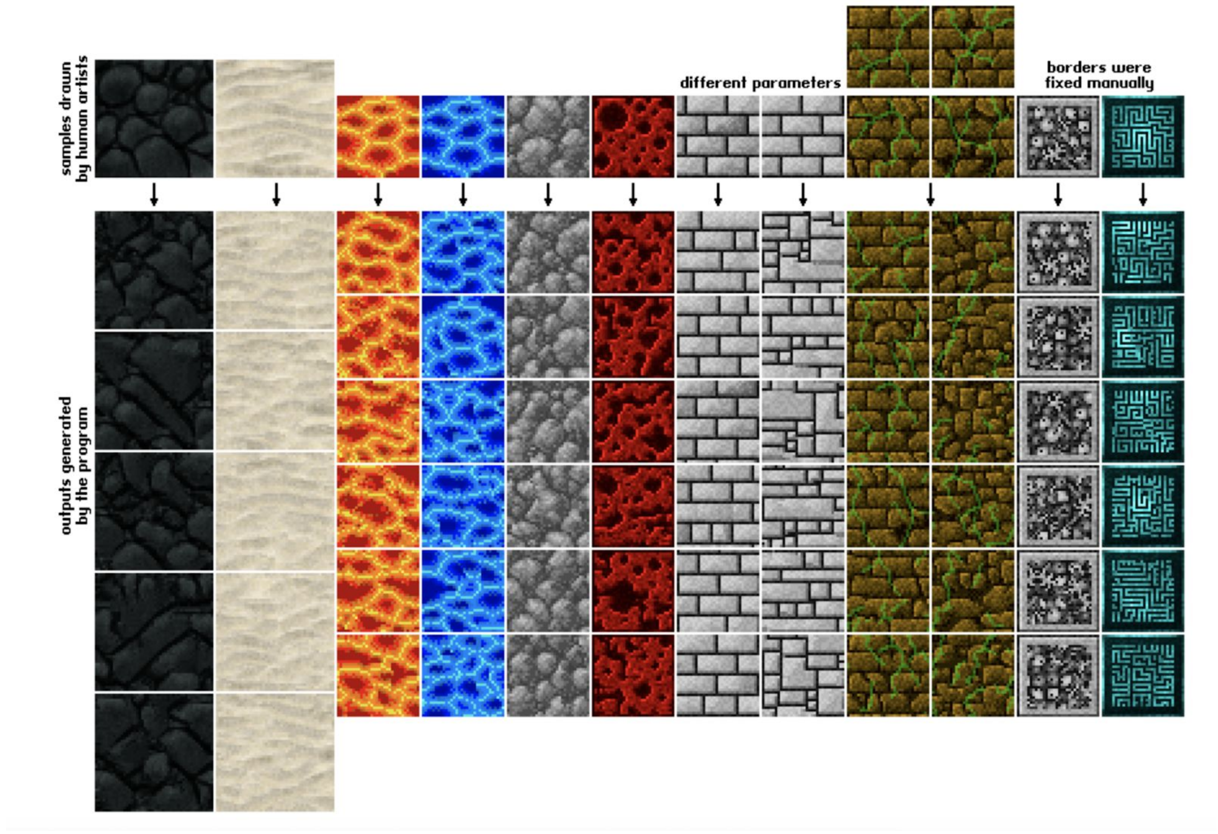
Cliff Zhu (zhaoxiz), Rosie Sun (weijias1)

Project Overview

In computer graphics, texture synthesis is an image-generation technique commonly used to produce high-quality textures when rendering synthetic images. The goal of our project is to use OpenMP to parallelize an algorithm called **non-parametric sampling** for texture synthesis.

Background

Define texture as some visual pattern of an infinite 2D plane. Texture synthesis is the process of taking a finite fixed shape sample from a texture to generate other samples from the given texture. Potential graphic applications of texture synthesis include image de-noising, occlusion fill-in, image compression, etc. Below are some examples of inputs and respective sample outputs of texture synthesis.

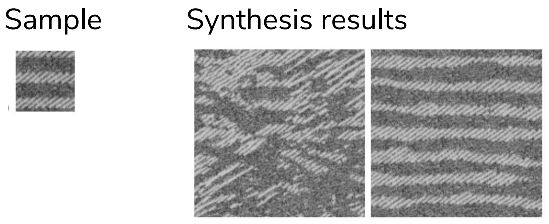

samples drawn by human artists
different parameters
borders were fixed manually
outputs generated by the program

Algorithms:

We present a high-level description below of the sequential algorithm based on the paper **Texture Synthesis by Non-parametric Sampling** by Alexei A. Efros and Thomas K. Leung of UC Berkeley.

Define:

- Neighborhood of p, ***ω*(p),** to be a NxN square window centered at pixel p
    - The size of the window N is a free parameter that allows the user to specify how stochastic the texture should be. Below the result on the left is generated with a small window size N that gives a high stochastic image, while a larger value of N yields a more uniformed synthesis on the right.



- **d(*ω*1, *ω*2)** to be the perceptual distance between any two neighborhoods. Smaller distance means that two neighborhoods have higher similarity.
    - To find the distance, the normalized sum of squared differences (SSD) approach is taken with slight optimization. The SSD approach gives the same weight any mismatched pixels regardless of their location. Since we would like to preserve the local structure of the original texture, a Gaussian kernel is used to assign less error weight to the edge pixels.

Algorithm:

```
For each pixel p in image:
      For all neighborhoods ω in the given sample:
         ●  Compute d(ω, ω(p)) based on modified SSD approach
         ●  Let ω_min to be the neighborhood with minimal distance
            (highest similarity) to ω(p).
         ●  Let S = { ω: d(ω, ω(p)) <= (1+ε)*ω_min } be the set of
            neighborhoods in sample with similarity above some threshold.
      Randomly select a neighborhood ω’ from S.
      Color current pixel p with the color of the center pixel of ω’
```

Challenges

When rendering images of large dimensions or when the sample pattern is fairly complex, the algorithm could take a long time to perform synthesis and generate the output image. After studying the algorithm, we also observed a few challenges to parallelize the sequential version of the algorithm:

Data dependency:
To compute the SSD to find the distance between a neighborhood in sample and the current neighborhood in output, we only want to include pixels that have been processed in the output. Here we encounter a very similar problem to computing rat moves in assignment 3, since the next iteration of computation will be relying on previously updated pixel values in the output. We plan to process pixels in batches to yield a smooth texture while optimizing the performance using parallelism.

Bad cache performance:
One potential source of parallelism is to parallelize over pixels to locate the neighborhood in the given sample for finding the source pixel color. If multiple pixels are trying to access the same neighborhood, it could affect cache performance due to false sharing of cache lines. This means we need to carefully design how to arrange data in memory to reduce false sharing.

Non-uniform Access Pattern:
In order to find a pixel to synthesize from, we need to examine all potential neighborhoods in the sample. As described above, the matched neighborhood in the sample is **randomly chosen** from a set of neighborhoods whose similarity to $\boldsymbol{\omega}$**(p)** is above some minimal threshold. Since the target neighborhood is selected randomly, access to values in pixel array will likely be non-uniform. This could lead to bad performance when accessing the memory.

Resources

Publications:
Efros, Alexei A., and Thomas K. Leung. "Texture synthesis by non-parametric sampling." Proceedings of the seventh IEEE international conference on computer vision. Vol. 2. IEEE, 1999.
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/papers/efros-iccv99.pdf

Wei, Li-Yi, and Marc Levoy. "Fast texture synthesis using tree-structured vector quantization." Proceedings of the 27th annual conference on Computer graphics and interactive techniques. 2000.
https://graphics.stanford.edu/papers/texture-synthesis-sig00/texture.pdf

Implementations
https://github.com/mxgmn/TextureSynthesis
https://github.com/thuliu-yt16/TextureSynthesis

Goals and Deliverables

Plan to Achieve:
Implement the sequential version of the algorithm described above.
Speed up over the sequential version using OpenMP and analyze its performance.

Hope to Achieve:
Speed up over the sequential version using OpenMP/CUDA, perform analysis and compare the parallized performances of the CUDA/OpenMP implementation.

Planned Final Deliverables:
A thorough analysis of performance comparisons between our parallel and sequential versions of texture synthesis with non-parametric sampling, with results of optimization from the benchmark experiments.

Platform Dependencies

C++, OpenMP, CUDA

Project Timeline (Proposed)

Date	**Milestone**
Apr. 5	Brainstorming ideas and finalizing project domain, propose timeline
Apr. 12	Conduct literature review Understand the algorithm to exploit potential parallelism Start working on the sequential implementation
Apr. 19	Finish sequential implementation and benchmark experiments
Apr. 26	Finish parallel implementation with OpenMP with performance measurement
May 1	Finish CUDA implementation and finalize analysis
May 4	Draft and submit final report