Course Project Report

Eduard Zalyaev
DS2-2016
Innopolis University
Innopolis, Russia
e.zalyaev@innopolis.ru

I. Introduction

During the development of a portfolio asset I created a basic colony simulator populated with agents that can distribute and perform simple tasks. Current build also supports a simple tile-based terrain renderer. However the quality of generated terrains is something I wanted to improve upon. Previously generated structures were repetitive and looked like they were created by a set of predictable patterns. In order to fix this I extended the base terrain generator with biomes and plants and also added smooth tile transitions and simple raytracing. It has to be noted that all evaluations related to both rendering and generation algorithms are stated in the portfolio elements.

II. Tiled Diamond Square

Diamond square [2] algorithm is generally used to produce realistic terrain heightmaps. It consists of two operations:

- **Square step**: set value of the square midpoint to average of it's corners
- **Diamond step**; set value of the diamond corners with an average value of it's center and two closest square corners. A random displacement value is also added to the average in order to increase noise.

In order to save GPU memory and CPU time I decided that the whole terrain has to be generated in chunks. That makes the generator more scalable and we can offload the generation to other threads. However the original implementation of the diamond square algorithm does not allow us to generate multiple segments that would seamlessly blend into each other. This means we have to generate the whole map in one go. In order to fix this problem the diamond step of the algorithm was changed so that instead of averaging only three values at the edges of the current chunk we would query the fourth one from another chunk.

This approach works quite well most of the time but has some issues in rare cases.

III. Biome Subdivisions

Addition of biomes required some updates to the tile atlas and rendering implementation as well as development of biome placement algorithm. The most obvious choice was Voronoi diagram [1] with randomly placed centers. Current implementation's chunks contain the coordinates and the biome type for 4 regions within it. During the assignment of biomes each tile would have to check all region coordinates in order to find the closest one. If we're generating infinite

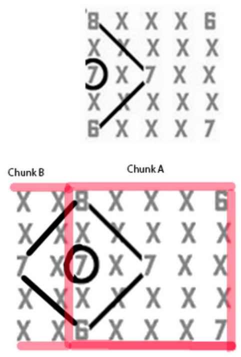

Fig. 1. Example of the original diamond step on the top and the reimplementation on the bottom.

landscapes that may pose as a big issue. That's why we evaluate only 8 neighbouring regions of the region where the tile is currently in and potentially save a lot of time.

IV. Plants

Plants are conveniently rendered in the same way as the terrain tiles are. They have specific biome types and subtypes (grass, bush or tree) and are placed with biome specific restrictions using Perlin noise. For instance grassland can't have any plants grow on sand but it does not matter for the desert and snowy biomes. Rendered plants can also wiggle as if the wind was coming through the area. This effect is based of the Graveyard Keeper lead programmer blog [3] on visual effects for pixelart.

V. Tile transitions

Terrain produced by the previous rendering pipeline was looking flat and I decided that we have to make transitions between tiles with different heights somehow. The solution I came up with consists of three steps.

- Render heightmap into separate texture.
- Apply Sobel convolution operator to detect edges on the heightmap.
- Invert the detected edge map and set it as alpha channel of the terrain texture.

This algorithm currently produces transitions of 2 pixel width which can be further increased by providing heightmap with more details.

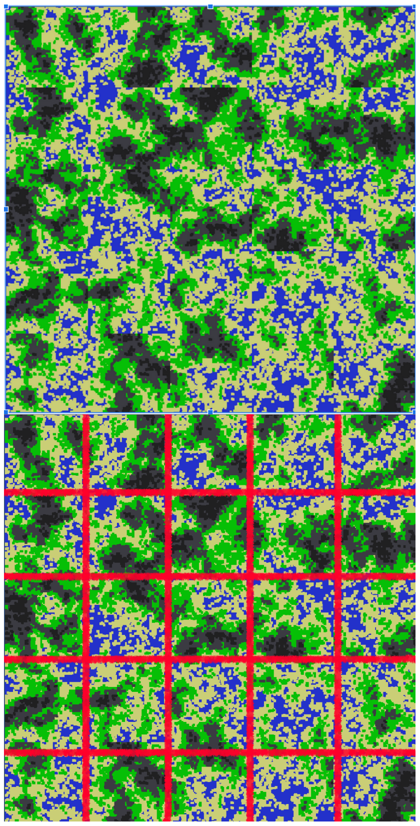

Fig. 2. Example of the produced results with tiling outlines

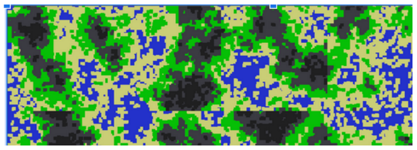

Fig. 3. Example of inconsistent tiling

VI. Lighting

One final step towards illusion of depth was to add some simple lighting to the scene. A simple raytracing algorithm was used to determine how occluded each area is from the sun. For every pixel A we start traversing the heightmap in the sun's direction sampling all the pixels underneath. If we encounter a pixel that has a higher height than A, we might have hit an occluder B. In order to determine if it is indeed occluded we have to do following steps:

- Cast a ray from A towards the sun, using sun's Z angle.
- Sample the ray's height at B
- If B's height is bigger than ray's height at this point, A is occluded

If A isn't occluded after n steps towards the light, it's considered to be unoccluded.

The results are pretty good however chunks do not share shadows with each other and rendering all this takes a lot of computing power. Possible solution to high GPU load might be updates with lower frequency or prebaked shadows.

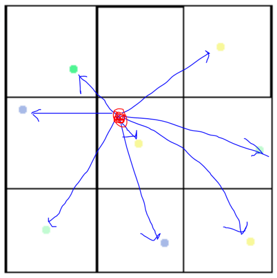

Fig. 4. Example of biome assignment for a single tile. Each square represents a region within the chunk and the colored points are biome centers

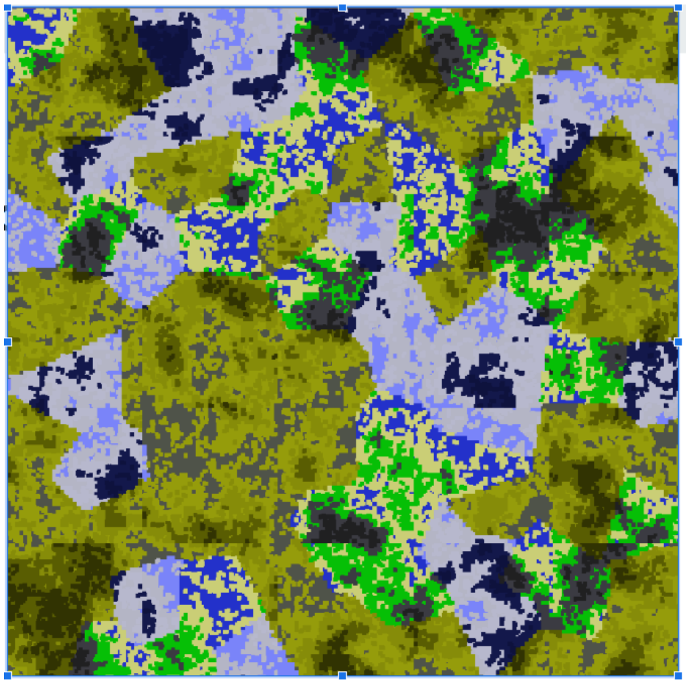

Fig. 5. Produced biome assignment. Grassland, desert and snowy biomes were added

References

[1] Voronoi diagram wiki page
[2] Diamond-Square wiki page
[3] Graveyard Keeper: How the graphics effects are made

*references are clickable

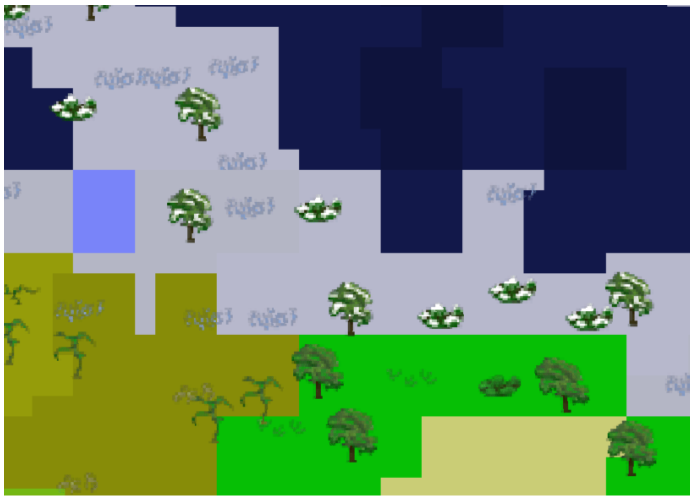
Fig. 6. Vegetation

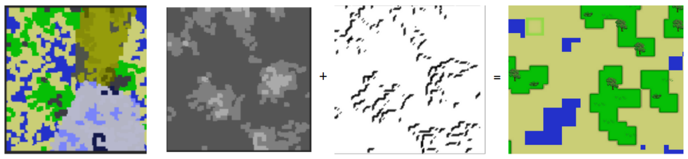
Fig. 7. Tile transition generation pipelie

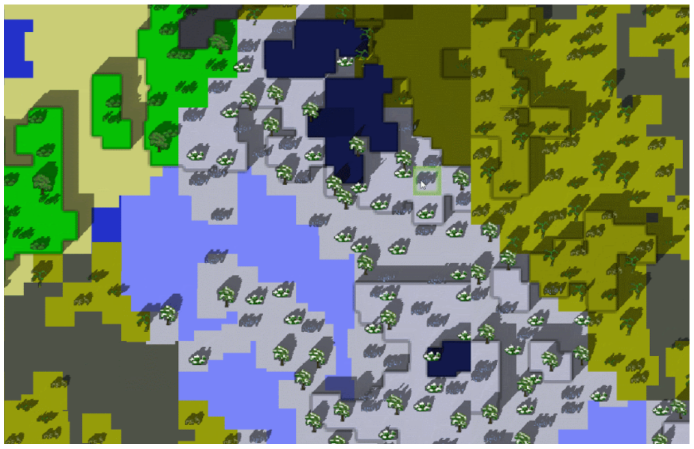
Fig. 8. Raytraced lighting results