Homework 1 - Height Field Intersection

Digital Image Synthesis

R05944045 陳卓晗

Date submitted: 10/27/2016

1. Heightfield intersection algorithm

In this project I used uniform grid to divide the whole space, and tried 3D-DDA and 2D-DDA algorithm for intersection. I will briefly talk about what I have done.
First I need to get the overall boundbox of the heightfield in order to **divide the space into uniform grid**. The number of partitions along the axis will be depend on the total number of primitives and the length of the axis with the largest extent. The grids along the axis will also be limited under 64 in case there are too many grids. After that I calculated each primitive's boundbox, then **added it's triIndex (index of the primitive) into the specific grid** once it touches one. So, when a ray hit the overall boundbox of the heightfield, it will **use 3D-DDA to traverse the grids along it's direction**. To check the intersect between a ray and a grid would also be expensive, luckily we can speed this step by making use of ***NextCrossingT***, which would be calculated before a ray is going to perform 3D-DDA. Then if the hitted grid was covered by a triangle's boundbox which was recorded previously, I will check if there is an intersection between ray and the triangles and return the foremost triangle the ray hitted.
My original code was really slow (more than 300s for landsea-big, compared to the built-in codes only took 21.3s), so I tried some different ways to optimize it including **improvement in data structure**, which helps a lot. But one big optimization is to **return the ray immediately when it hit a triangle**. Since the heightfield is so well organized, we don't need to worry whether there is another triangle closer to the eye. Both optimization makes my rendering quicker (from 300s to 117s in landsea-big), although it was still much slower than the built-in algorithm.
Since the 3D-DDA have much more grids to test along the z-axis, I also tried 2D-DDA algorithm. I lower the heightfield dimension to 2D and **only consider step change along x and y axis**. Because there are exactly 2 triangles in each heightfield grid, I also don't need to store the triIndex anymore. So when a ray move forward along it's direction, it will test the intersection with 2 triangles in each grid and return the tHit when it hit a triangle. To my surprise, this 2D-DDA was slightly slower than 3D-DDA, (from 37.3s to 59s in landsea-0). 2D-DDA reduces the grids it traverse, but increase the triangle intersection tests.

2. Phong shading interpolation

To calculate phong shading interpolation we need to **get the normal vector of every hit point**, it can be calculated by cross(dpdu, dpdv). The GetShadingGeometry() function will return the differential geometry of a "true triangle mesh" by default, which makes the land and sea looks flat. So we need to implement the interpolation there. Generally I **use the (H(x - 1) - H(x + 1)) / 2 to simulate the dpdu at point x** since heightfield doesn't have that parameter. For a point in a triangle, it's **dpdu and dpdv can be interpolated by it's 3 surrounding triangle points**. Then it's normal could be calculated through dpdu and dpdv. At first I used bilinear interpolation which makes the image a little flat, the output image became much better when I used barycentric interpolation. After GetShadingGeometry() returns the "true normal" of a point, the surface can be much more smooth.

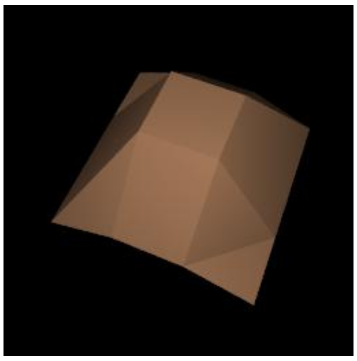
3. Performance

Environment	CPU	Memory	Cores	GPU
Windows 10	Core i7 2.70GHz	8GB	8	GTX 970m

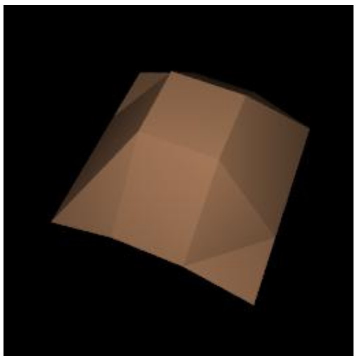
	Default	UG + 3D-DDA		UG + 3D-DDA /w phong	UG + 2D-DDA /w phong
hftest	3.2s	4.3s	130%	4.4s	3.1s
Landsea-0	18.3s	36.6s	200%	37.1s	63.0s
Landsea-1	10.3s	18.2s	176%	18.5s	21.3s
Landsea-2	8.7s	15.0s	172%	16.2s	24.3s
Landsea-big	21.7s	115.8s	533%	117.8s	356.3
texture	6.0s	12.9s	215%	13.2s	15.6s

4. Results

Default



Mine



Mine /w phong