设计模式文档

本软件是类似CAD的建模软件的初步实现。借助Unity开发且主要使用marchingCube算法以绘制体积模型。

首先是使用到了C2风格的设计模式。由于和用户交互的操作十分频繁，所以图形界面是必不可少的。

1. **public** **class** Mouse : MonoBehaviour
2. {
3. Camera m\_Camera = **null**;
4. **public** Vector3 size = **new** Vector3(0.5f, 0.5f, 0.5f);
5. **private** Vector3 position = **new** Vector3(0, 0, 0);
6. [SerializeField] GameObject cursor = **null**;
7. [SerializeField] MeshGenerator generator = **null**;
9. [SerializeField] Vector3 canvasSize = Vector3.one \* 16;
10. [SerializeField] Vector3Int canvasDensity = Vector3Int.one \* 64;
11. [SerializeField] **float** initSphereRadius = 10;
13. [SerializeField] **float** radius = 1;
14. [SerializeField] **float** strength = 1;
15. [SerializeField] **float** damping = 0;
17. **public** **bool** mirror\_x = **false**;
18. **public** **bool** mirror\_y = **false**;
19. **public** **bool** mirror\_z = **false**;
21. // Start is called before the first frame update
22. **void** Start()
23. {
24. m\_Camera = Camera.main;
25. m\_Camera.depthTextureMode = DepthTextureMode.None;
27. cursor = Instantiate(cursor);
28. cursor.transform.localScale = size;
29. cursor.transform.position = position;
30. cursor.SetActive(**false**);
32. Volume sphere = **new** Volume(canvasSize, canvasDensity);
33. generator.InitVolume(sphere, initSphereRadius);
35. generator.InitBuffers(sphere);
36. generator.ReadData(sphere);
37. generator.MarchAll();
38. Mesh mesh = generator.GenerateMesh();
39. generator.SetMesh(mesh);
40. }
42. // Update is called once per frame
43. **void** Update()
44. {
45. Ray ray = m\_Camera.ScreenPointToRay(Input.mousePosition);
46. **if** (cursor.activeSelf == **false**)
47. cursor.SetActive(**true**);
48. **if** (Physics.Raycast(ray, **out** RaycastHit hit))//击中mesh
49. {
50. position = hit.point;
51. cursor.transform.position = position;
52. cursor.transform.LookAt(position + hit.normal);
53. **if** (Input.GetMouseButton(0))
54. {
55. Mirror(hit.point, **out** Vector3[] pos);
56. **foreach**(Vector3 p **in** pos) Modify(p, radius);
57. Mesh mesh = generator.GenerateMesh();
58. generator.SetMesh(mesh);
59. }
60. }
61. }

以上是Mouse.cs中对用户鼠标射线的检测代码以及部分处理函数。

其次，由于借助Unity引擎开发，不得不在其逻辑结构中进行脚本的绑定和函数调用。而其逻辑结构是顺序式的类似于层次的按时间顺序的。比如生命周期中的Start和Update。

因此其中有层次化的设计模式，见上述代码。

再者，在计算着色器即ComputeShader中需要GPU和CPU之间的数据传递，那么当然就需要一个双缓冲来匹配两者的速度。这类似于设计模式中的仓库模式。

1. // Each #kernel tells which function to compile; you can have many kernels
2. #pragma kernel Modify
4. **static** **const** **int** numThreads = 8;
6. float3 **base**;
7. float3 voxelCount;
8. float3 center;
9. **float** radius;
11. **float** strength;
12. **float** damping;
14. **int** numPointsX;
15. **int** numPointsY;
16. **int** numPointsZ;
18. RWStructuredBuffer<float4> points;
20. **int** indexFromCoord(**int** x, **int** y, **int** z) {
21. **return** y \* numPointsY \* numPointsX + x \* numPointsX + z;
22. }
24. [numthreads(numThreads,numThreads,numThreads)]
25. **void** Modify (int3 id : SV\_DispatchThreadID)
26. {
27. int3 bas = int3((**int**)**base**.x, (**int**)**base**.y, (**int**)**base**.z);
29. float3 cid = id + bas + float3(1,1,1);
30. **if** (cid.x >= numPointsX-1 || cid.y >= numPointsY-1 || cid.z >= numPointsZ-1 || cid.x <= 0 || cid.y <= 0 || cid.z <= 0) **return**;
31. **if**( id.x >= voxelCount.x-1 || id.y >= voxelCount.y-1 || id.z >= voxelCount.z-1) **return**;
33. **int** index = indexFromCoord(cid.x, cid.y, cid.z);
35. float3 pos = points[index].xyz;
36. **float** value = points[index].w;
38. //float dis = clamp(length(pos - center), 0, radius);
39. **float** dis = length(pos - center);
40. **float** weight = clamp(dis / radius,0,1);
42. **if**(dis > radius) **return**;
44. **float** delta = lerp(strength, strength \* (1 - damping), weight);
46. value = clamp(value + delta, -256, 256);
47. //value = value + delta;
49. points[index].w = value;
50. }