目录

RayMarching and Signed Distance Functions

Lei Xinyue

参考:[1]

1 Signed Distance Functions

Signed Distance Functions，简称为 SDF，有符号距离函数。给定空间中一个点 P 的坐标，SDF 会返回从 P 到某个曲面 (surface) 的最短距离 (可能为正数，也可能为负数)。(正负) 符号表示点 P 是在曲面内还是外。

例如：有一个球，球心在 C 点，半径为 r。对于在球内的任一点 P，其到球心 (也即原心) 的距离小于半径；若点在球面上，则距离等于半径；若点在球外，则距离大于半径。其对应的 SDF 为:

$$f(\vec{P}) = \left\| \overrightarrow{PC} \right\| - r$$

1.1 Primitives

1.1.1 Sphere

1.1.2 Box

2 The Raymarching Algorithm

一旦一个模型以 SDF 的形式给出，可以使用 Raymarching 算法渲染它。和光线追踪算法类似 (图1)，选择一点作为相机 (Camera) 的位置，在相机前面放一个虚拟“网格”(视平面，相当于屏幕)，从相机发出一条条穿过网格的视线 (View Ray)。每一个网格格点对应输出图像中的一个像素。在 Raymarching 中，整个场景是使用 SDF 定义的。为了找到视线与场景的交点，可以从相机处 (视线原点) 开始，沿着视线，一点一点 (a very small increment) 地移动一个点 (a point)。每移动一次，就问“当前点在场景曲

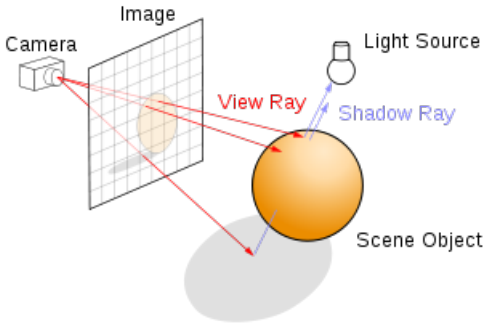

图 1: 光线追踪示意图

面 (scene surface) 内部么?”，即以当前点为输入，SDF 返回值是否为负值。如果为负，则视线与场景发生了碰撞；否则，继续沿着视线移动点，直到达到最大步数。

上面这段描述的” 一点一点” 地移动点的方法，可能需要迭代很多次才能发现视线有没有碰撞到场景。可以使用”sphere tracing” 的方法，进行改善。每次沿着射线移动最大的全距离 (保证不穿过场景曲面): 由 SDF 提供的当前点到场景曲面的距离。

如图2，p_0 是相机处 (视线原点)。蓝色的线是视线的一部分。从 P_0 慢慢到 P_4，每次移动的距离是球 (示意图中的圆) 的半径。如在 p_0 处，发现到场景曲面的最近点不在视线上，就继续沿视线向前移动，移动距离为此时 p_0 到场景的距离。

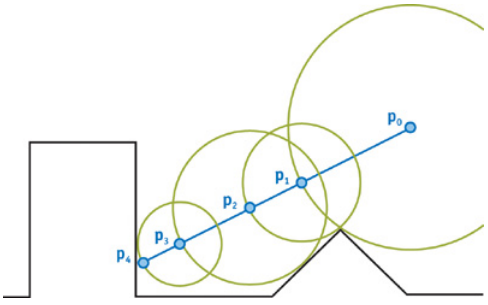

图 2: spheretracing 示意图

如下代码可以在shadertoy中运行，使用 raymarching 渲染出一个球:

```
1
```

```
2  const int MAX_MARCHING_STEPS = 255;
3  const float MIN_DIST = 0.0;
4  const float MAX_DIST = 100.0;
5  const float EPSILON = 0.0001;
6
7  /**
8   * 球心中原点,
9   半径为1的球的Signed distance function
10  */
11 float sphereSDF(vec3 samplePoint) {
12     return length(samplePoint) - 1.;
13 }
14
15 /**
16  * 描述场景的SDF
17  * 返回值是点samplePoint到场景的最短距离,
18  * 若为负值表示 * 该点在场景曲面内
19  * (即与场景发生了碰撞)
20  */
21 float sceneSDF(vec3 samplePoint) {
22     return sphereSDF(samplePoint);
23 }
24
25 /**
26  * 返回视点(相机处), 沿着视线(marching方向),
27  * 达到场景的最短距离。
28  * 如果没有找到场景曲面则返回end.
29  *
30  * eye: 相机位置, 视线原点
31  * marchingDirection: marching方向,
32  即视线方向, 单位向量
33  * start: 从相机开始的起始移动距离
34  * end: 最大距离(放弃的移动)
```

```
35  */
36 float shortestDistanceToSurface(vec3 eye,
37  vec3 marchingDirection, float start, float end) {
38     float depth = start;
39     for (int i = 0; i < MAX_MARCHING_STEPS; i++) {
40         float dist = sceneSDF(eye + depth * marchingDirection);
41         if (dist < EPSILON) {
42                         return depth;
43         }
44         depth += dist;
45         if (depth >= end) {
46             return end;
47         }
48     }
49     return end;
50 }
51
52
53 /**
54  * 给定屏幕(输出图像或视平面)上的一个像素,
55  返回对应的raymarching方向, 即视线方向。
56  *
57  * fieldOfView: 竖起方向的视角
58  * size: 输出图像的分辨率
59  * fragCoord: 输出图像中的一个像素点的坐标
60  */
61 vec3 rayDirection(float fieldOfView, vec2 size,
62   vec2 fragCoord) {
63     vec2 xy = fragCoord - size / 2.0;
64     float z = size.y / tan(radians(fieldOfView) / 2.0);
65     return normalize(vec3(xy, -z));
66 }
67
```

```
68
69 void mainImage( out vec4 fragColor, in vec2 fragCoord )
70 {
71         vec3 dir = rayDirection(45.0, iResolution.xy, fragCoord);
72     vec3 eye = vec3(0.0, 0.0, 5.0);
73     float dist = shortestDistanceToSurface(eye, dir, MIN_DIST, MAX_DIST);
74
75     if (dist > MAX_DIST - EPSILON) {
76         // Didn't hit anything
77         fragColor = vec4(0.0, 0.0, 0.0, 0.0);
78                 return;
79     }
80
81     fragColor = vec4(1.0, 0.0, 0.0, 1.0);
82 }
```

3 Surface Normals and Lighting

参考文献

[1] http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/

[2] http://www.codinglabs.net/article_world_view_projection_matrix.aspx

[3] http://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/how-does-it-work