创建球体网络的四种方法

翻译:fangcun

2018 年 10 月 23 日

目录

1 引言 **3**

2 构造网络 **3**

2.1 Standard Sphere 3
2.2 Normalized Cube 4
2.3 Spherified Cube 4
2.4 Icosahedron 5

3 构造方法比较标准 **5**

3.1 结果 5
3.1.1 网络到球体的距离: 平均误差 6
3.1.2 网络到球体的距离: 最大误差 7
3.1.3 每个三角形的面积: 均方误差 8
3.1.4 每个三角形的面积: 最大平方误差 9
3.1.5 表面距离误差的视觉表现 10
3.2 注释 10
3.3 总结 11

1 引言

本文介绍四种构建球体的方法，并对它们进行比较，来使大家可以从中挑选到适合自己的方案进行球体构造。

2 构造网络

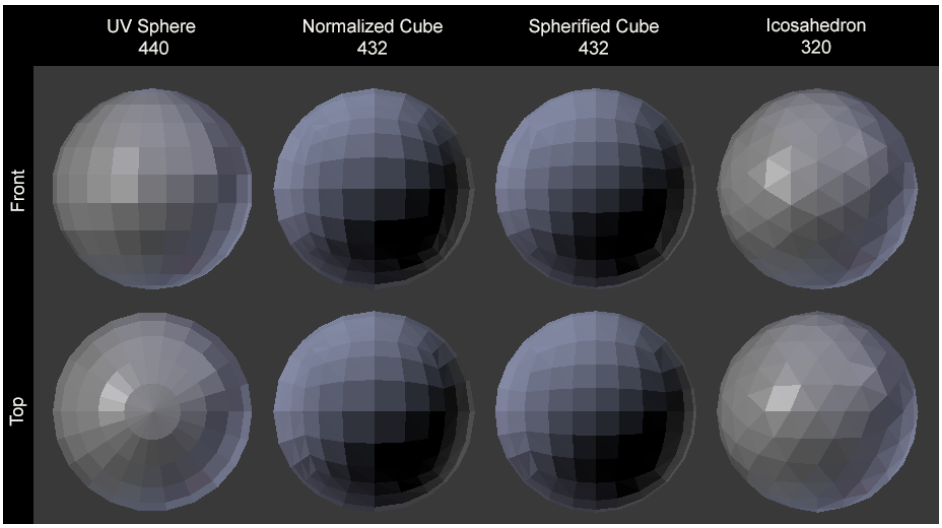


图 1: 四种不同方法构造出的球体效果

2.1 Standard Sphere

这是最常见的出现在 3D 软件中的构造球体的方法。在 **Blender**中，它被叫做 *UV Sphere*，在 **3d max**中，它直接被叫做 *Sphere*。这个方法使用经纬线[1]来构造球体，它在赤道附近会产生较大面积的单个细分表面，在极点附近相反。

```
for j in parallels_count:
        parallel = PI * (j+1) / parallels_count
        for i in meridians_count:
                meridian = 2.0 * PI * i /meridians_count
                return spherical_to_cartesian(meridian,parallel)
```

[1]从极点出发结束于极点和平行于赤道的线段。

2.2 Normalized Cube

这个方法使用一个立方体，对这个立方体进行细分得到顶点，然后把这些进行规范化，最后乘以球体半径得到新的顶点位置。这个方法构造的球体在立方体表面的中心会有较大的面积。

```
for f in faces:
        origin = get_origin(f)
        right = get_right_dir(f)
        up = get_up_dir(f)
        for j in subdiv_count:
                for i in subdiv_count:
                        face_point = origin + 2.0 *
                                (right * i + up * j)
                                        / subdiv_count
                        return normalize(face_point)
```

2.3 Spherified Cube

这个方法同样使用立方体进行，但它可以得到面积和边长变化更小的细分。越接近立方体一角的地方变形越明显。

```
for f in faces:
        origin = get_origin(f)
        right = get_right_dir(f)
        up = get_up_dir(f)
        for j in subdiv_count:
                for i in subdiv_count:
                        p = origin + 2.0 * (right * i + up * j) /
                                subdiv_count
                        p2 = p * p
                        rx = sqrt(1.0 - 0.5 * (p2.y + p2.z)
                                + p2.y*p2.z/3.0)
                        ry = sqrt(1.0 - 0.5 * (p2.z + p2.x)
                                + p2.z*p2.x/3.0)
                        rz = sqrt(1.0 - 0.5 * (p2.x + p2.y)
                                + p2.x*p2.y/3.0)
                        return (rx, ry, rz)
```

2.4 Icosahedron

正二十面体是由 20 个等边三角形构成的多面体，它有一些有用的性质: 每个三角形具有相同的面积，每个顶点到它相邻顶点的距离相同。

我们可以通过将其中一个三角形细分为四个来进行细分[2]。然而，这使正二十面体的性质被破坏掉，三角形不再使等边的，面积相等的，相邻顶点间的具体也不再相同。还有一个问题时，这个方法一次只能增加四倍的细分表面。但排除掉三角形的数量，这个方法仍然是一个不错的近似方法。

由于它的初始 12 个顶点和 20 个面需要手工编写，太过冗长，所以，这里没有给出它的伪代码。细分的伪代码在下面给出：

```
for f in input_mesh.faces:
        v0 = input_mesh.vertex(f, 0)
        v1 = input_mesh.vertex(f, 1)
        v2 = input_mesh.vertex(f, 2)
        v3 = normalize(0.5 * (v0 + v1))
        v4 = normalize(0.5 * (v1 + v2))
        v5 = normalize(0.5 * (v2 + v0))
        output_mesh.add_face(v0, v3, v5)
        output_mesh.add_face(v3, v1, v4)
        output_mesh.add_face(v5, v4, v2)
        output_mesh.add_face(v3, v4, v5)
```

3 构造方法比较标准

使用三角形网络近似表示球体不可能完美，这里，我们使用球体上的一些点到网络的距离作为标准来评价创建的网格。

另一个评价标准时期望三角形面积和实际三角形面积之比。它通常对细分更统一的三角形时很重要。

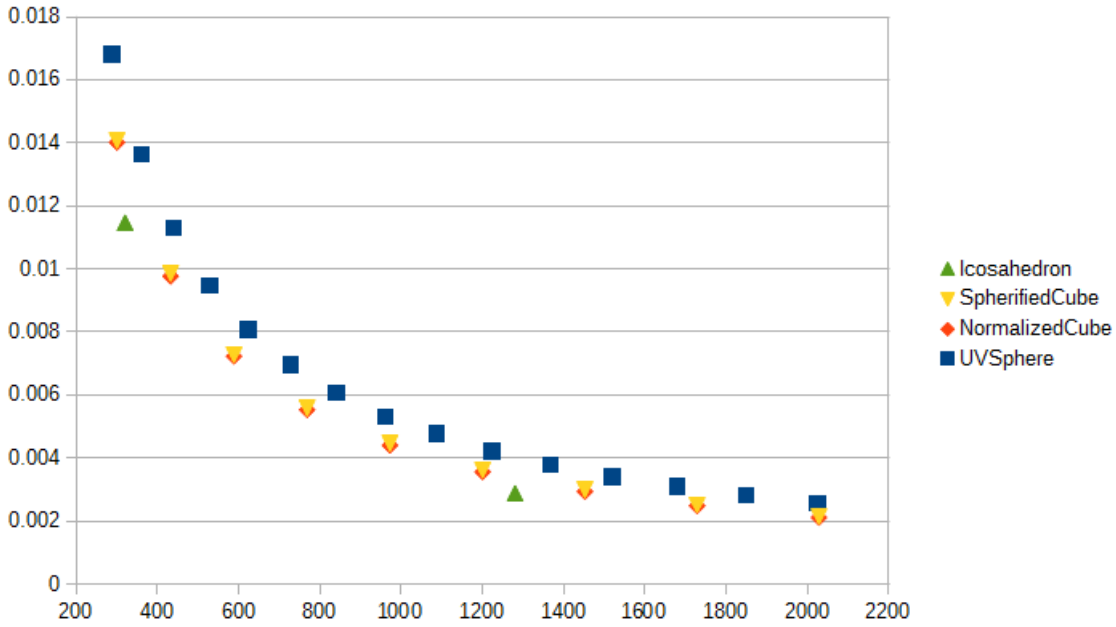
所有评价标准都给出了最大误差和均方误差，这些值越小表示生成的网络越好。

3.1 结果

表格的 x 轴表示三角形数量，y 轴表示误差。

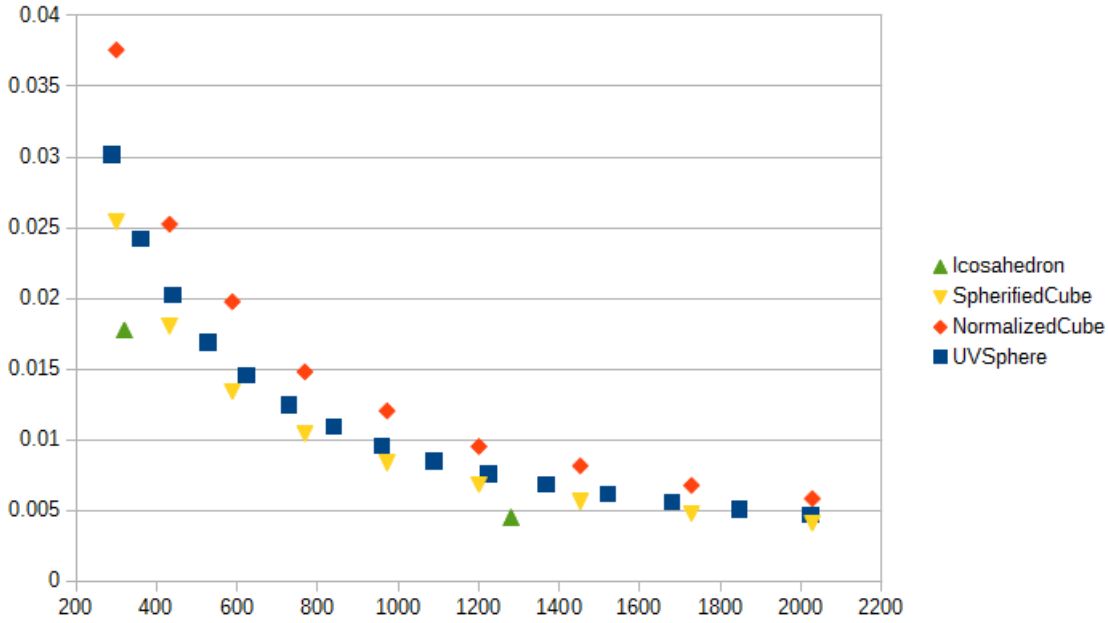
[2]具体是在三角形的每条边的中点创建新的顶点，然后对它进行规范化，使它落在球体表面。

3.1.1 网络到球体的距离: 平均误差



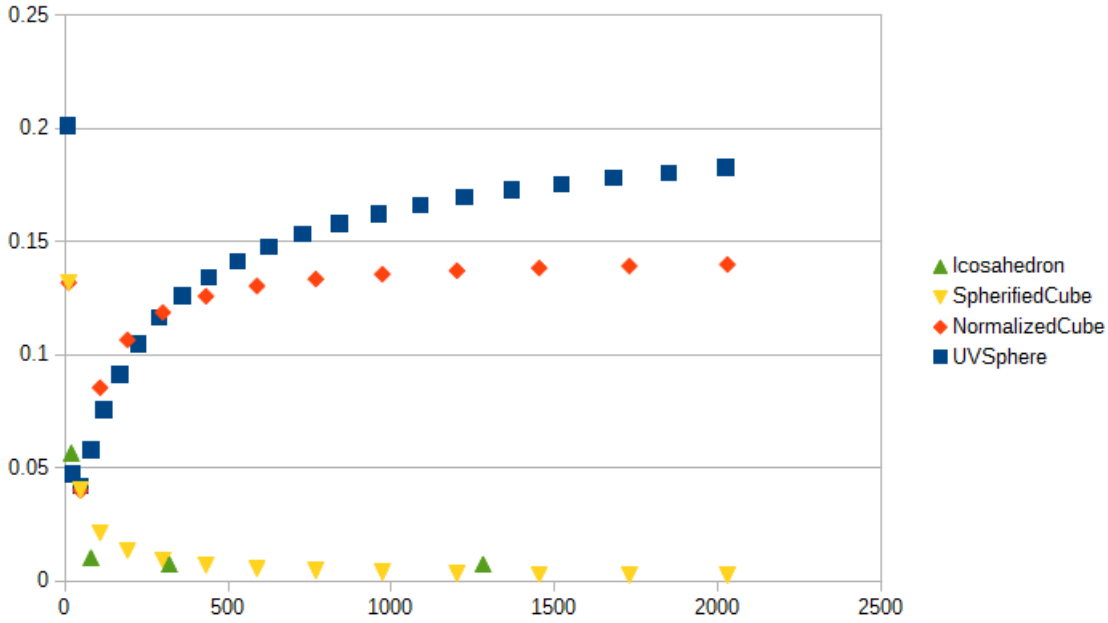
我们可以看到 **standard sphere**具有最大的误差，**icosahedron**的误差要比其它方法都小，但是对于我们这里分析的三角形数量，我们只能使用 320 和 1280 这两个三角形数量，这大大限制了这个方法的灵活性。

3.1.2 网络到球体的距离: 最大误差



normalized cube和 **spherified cube**的平均误差十分接近，但 **normalized cube**的最大误差较大，甚至大于 **uv sphere**。**icosahedron**仍然表现最好。

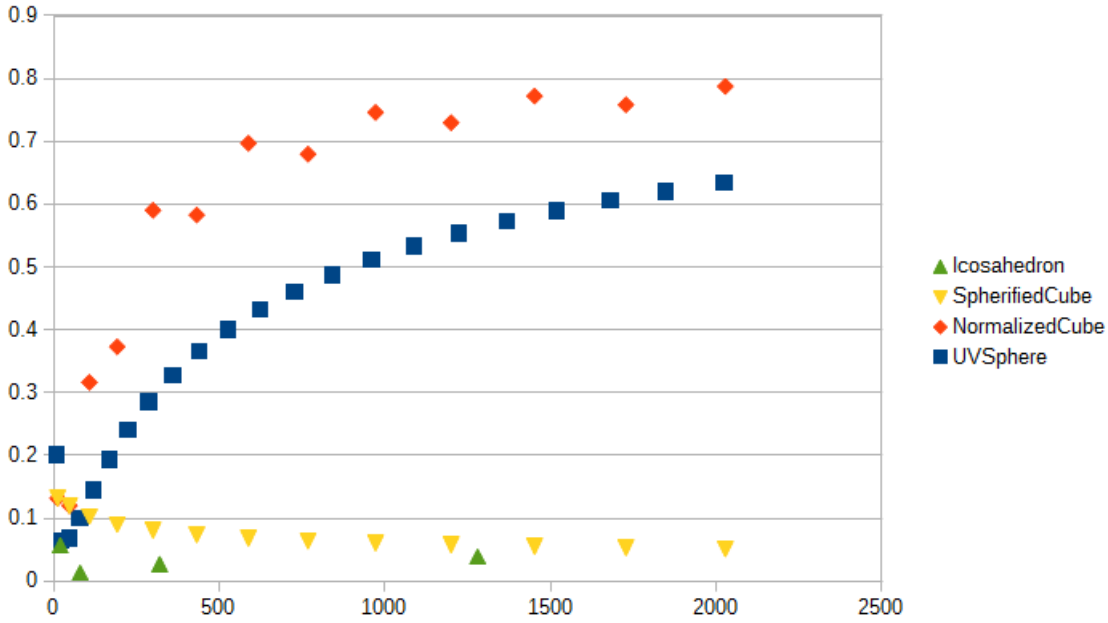
3.1.3 每个三角形的面积: 均方误差



这里我们可以看到 **spherified cube**的好处。它是网络三角形面积和期望三角形面积 [3]之比表现最好的方法，随着三角形数量的增加，它的表现越好。由于生成的新的四个三角形的面积的些许不同导致 **icosahedron**的误差增加。

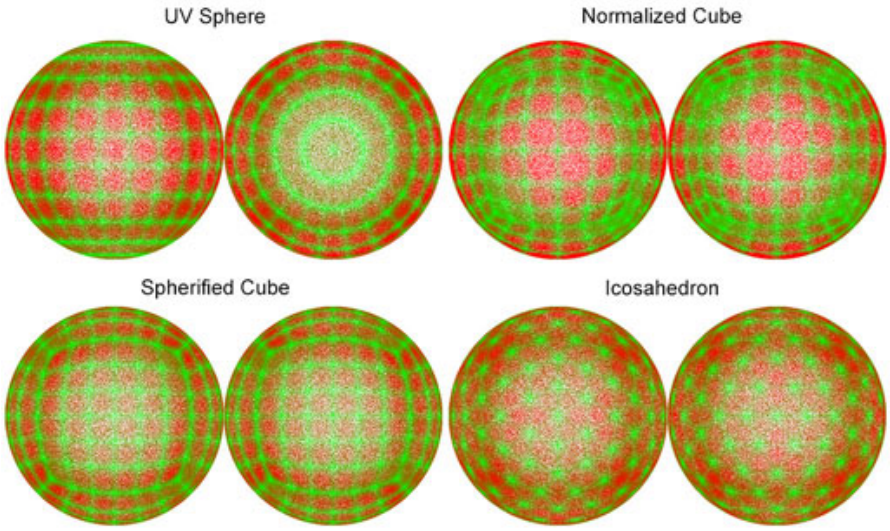
[3]球体面积处以网络中的三角形个数。

3.1.4 每个三角形的面积: 最大平方误差



我们可以看到面积的最大平方误差和到表面的最大误差类似。对于**平均误差，normalized cube**好于 **UV sphere**，但对于最大误差则相反。最大平方误差 **icosahedron**要比 **spherified cube**小，但我们可以看到随着细分三角形的增加，这个有增大的趋势。

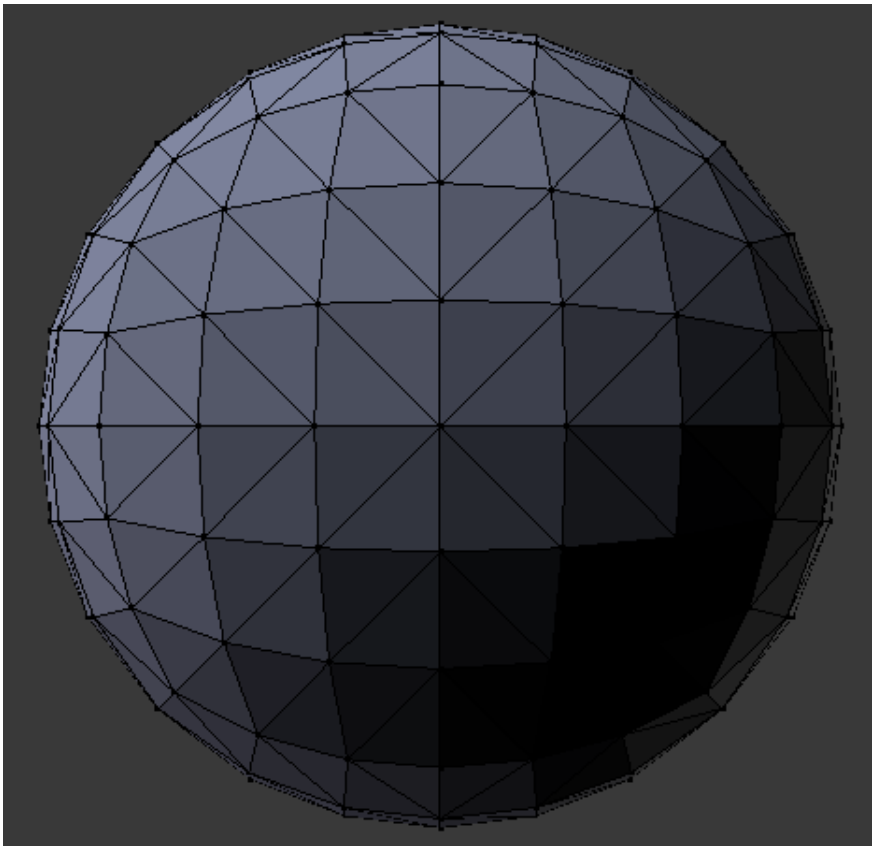
3.1.5 表面距离误差的视觉表现



3.2 注释

这个实现会在立方体的边上产生重复顶点。可以使用更复杂的代码避免。

可以通过对称的方法来提升细分立方体的三角形划分。



3.3 总结

UV sphere构造的球体是最差的，但它的算法是最简单的。如果你需要更好的构造球体，可以选择其它算法。