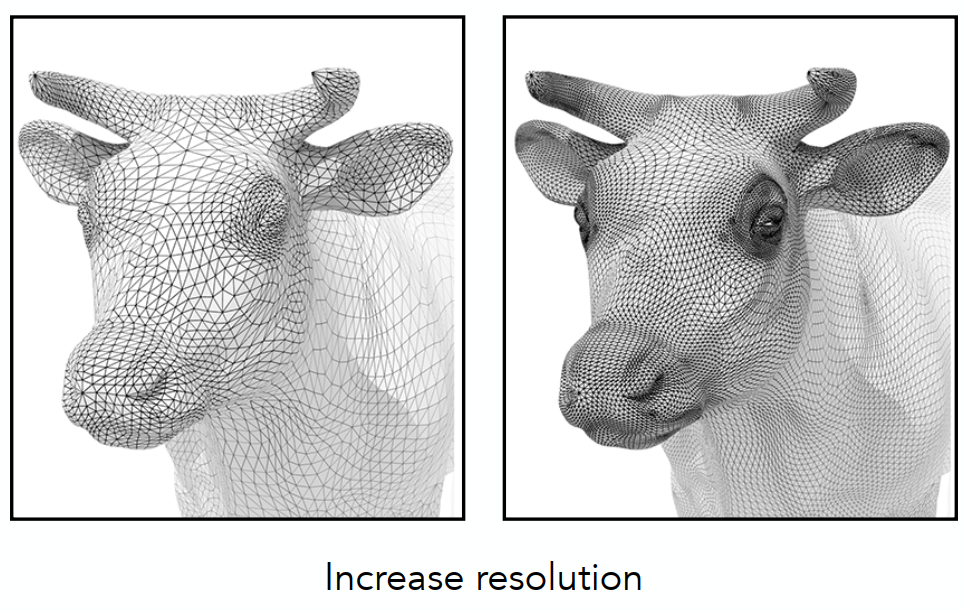
# Mesh Operations: Geometry Processing

Mesh subdivision

Mesh simplification

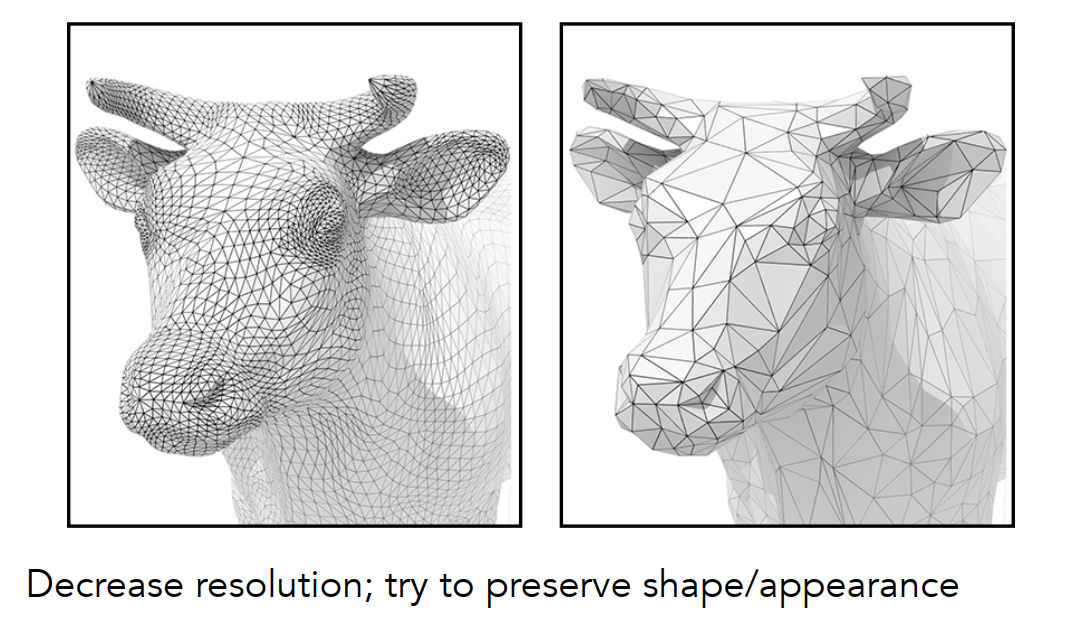
Mesh regularization

### Mesh Subdivision (upsampling)



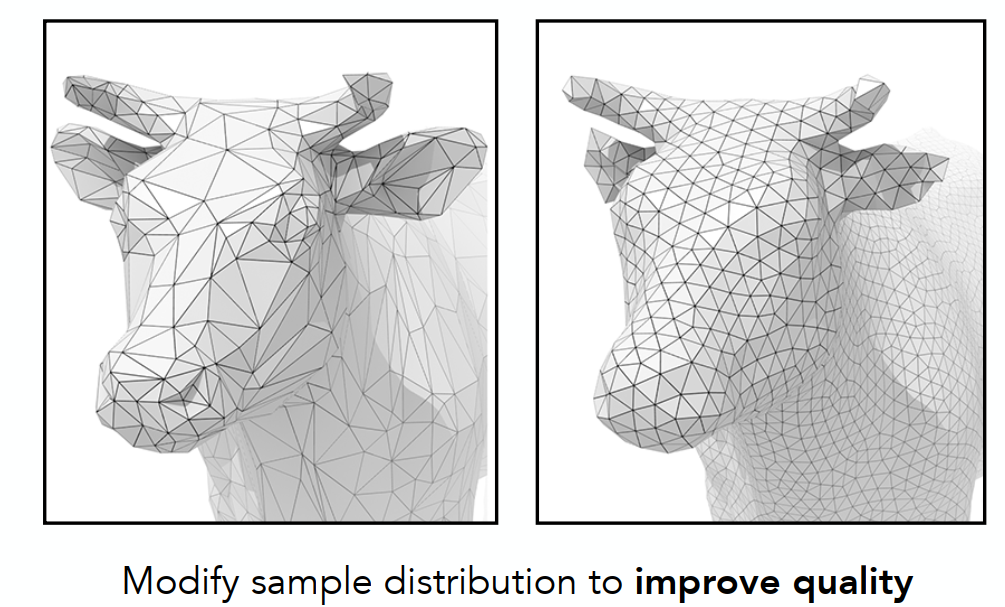
网格细分。通过增加三角形面数来得到更加平滑的效果。

### ②Mesh Simplification (downsampling)



网格简化。当一个模型很复杂，但是我看这个模型的时候距离相距很远，那么看到的细节其实没有那么多，这时候就需要对网格进行简化，去掉一些边、三角形等，但是依然要维持原来的相互连接关系，不能破坏基本结构。

③Mesh Regularization (same #triangles)



三角形大小不一，长短各不相同的模型，在渲染上可能会造成很多不便。这里做一个正则化，将原来的三角形正则成大小相同的正三角形，但是同时不能丢失细节。

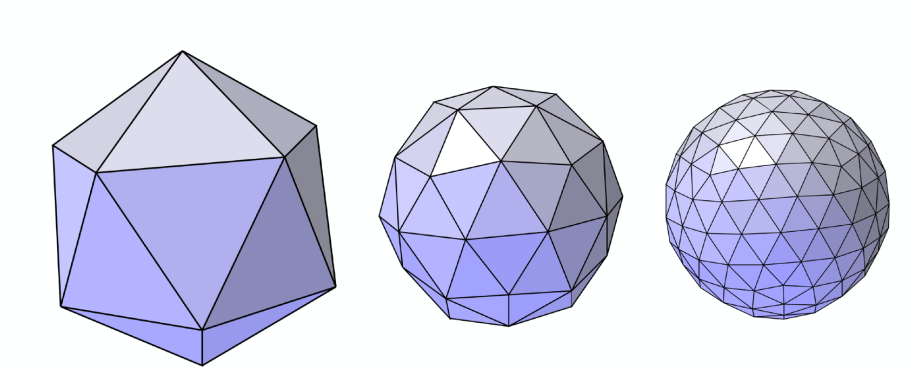
# (一)Subdivision细分

## 1、Loop Subdivision(注意发明者就叫loop 这个算法和循环没什么关系)

Common subdivision rule for triangle meshes

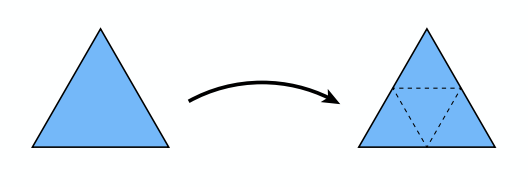
--First, create more triangles (vertices)

--Second, tune(调整) their positions



整个Loop细分分为两步，首先引入更多的三角形，然后再通过改变这些三角形顶点的位置来让整个模型看上去更光滑。

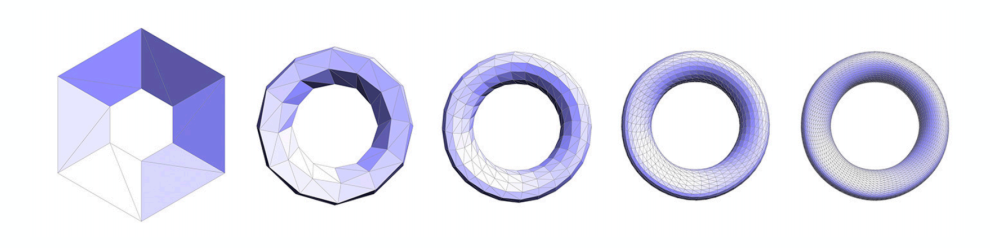
### Split each triangle into four



取三角形三边的中点并相互连接，就将一个三角形分成了四个三角形

Assign new vertex positions according to weights

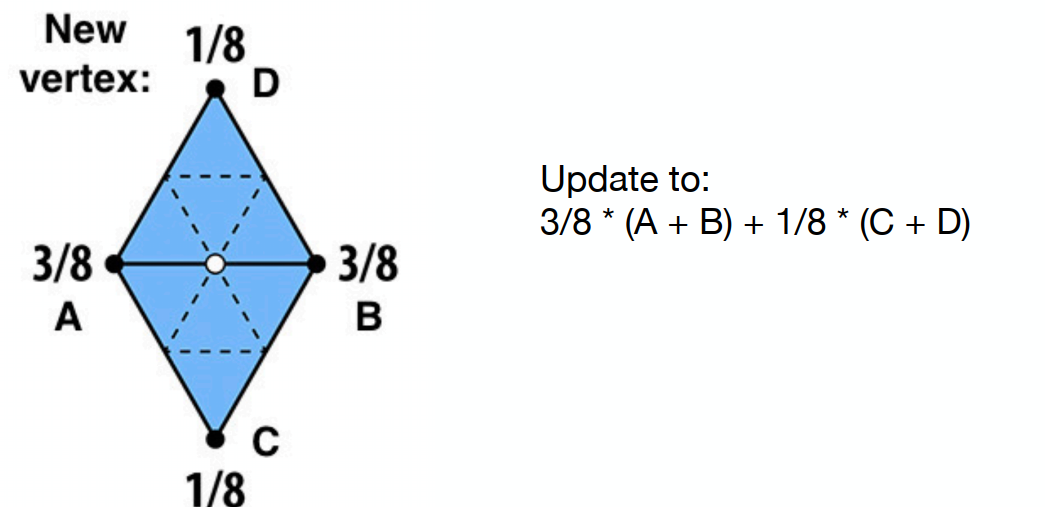
​ --New / old vertices updated differently根据权重加权顶点位置，使得螺母光滑成圆环



### Tune their positions

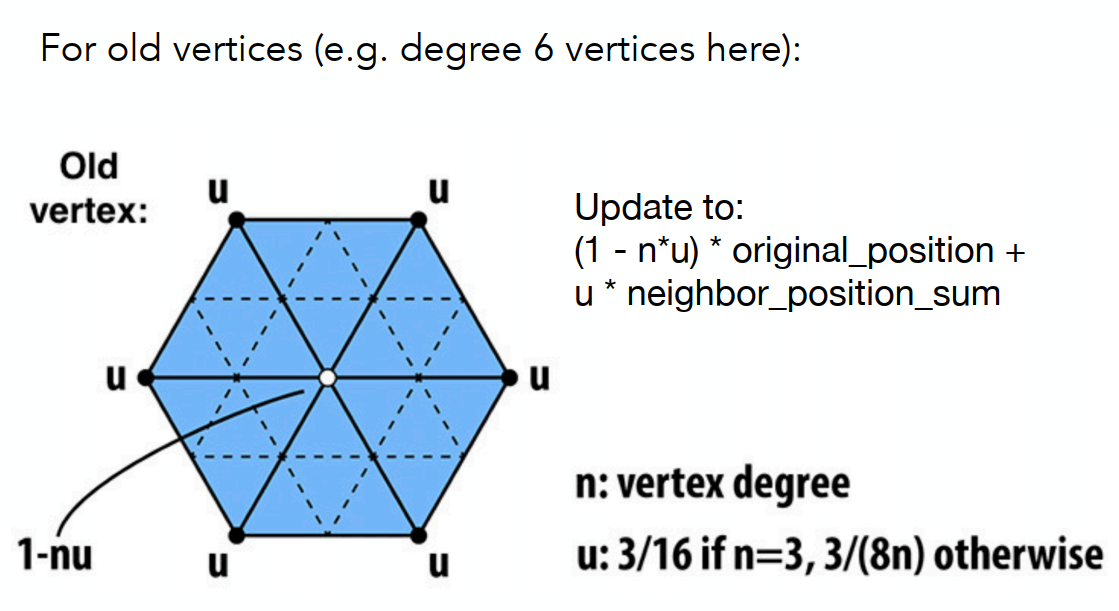
将三角形的顶点区分为新的顶点和老的顶点，新的顶点即为新生成的顶点（上一步中取到的边的中点），老的顶点即为原来三角形的顶点。Loop细分将这两种不同类型的顶点，分别以不同的规则来改变他们的位置。

#### For new vertices：

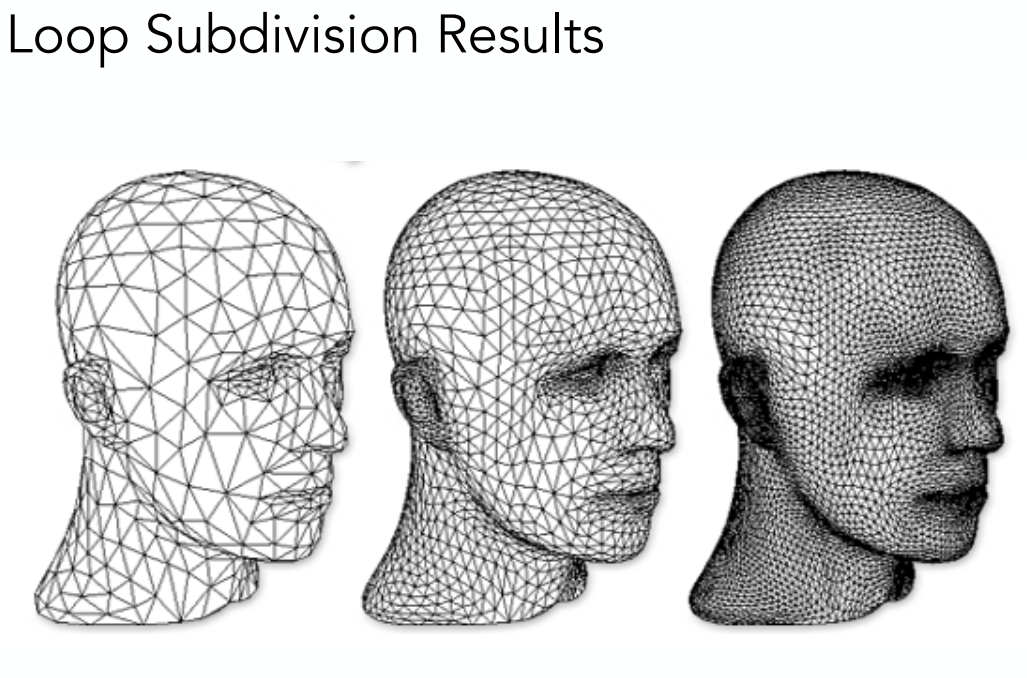


对于新的顶点（如上图白色点），会被两个三角形所共享，将共享顶点所在的这条公共边两端的两个公共顶点分别定义为A、B，然后把这两个三角形非公共的两个顶点分别定义为C、D，然后将共享顶点（上图白色点）的位置调整为3/8 \* (A + B) + 1/8 \* (C + D)即可（这个式子其实就是一个简单的加权平均，A和B离共享顶点近一些，所以占的比例相对较大，而C和D离共享顶点远一些，所以占的比例相对较小）。

##### For old vertices (e.g. degree 6 vertices here)

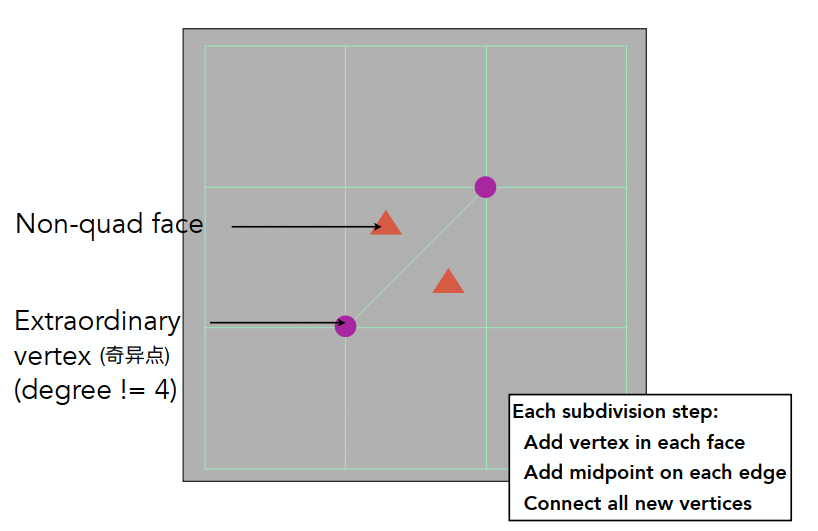


对于老的顶点（如上图白色点），周围的老顶点会对这个顶点有一定的影响。Loop细分规定这个要计算的老顶点（上图白色点）一部分受周围老顶点的影响，一部分保持自己的位置属性。在这里我们定义一个n，为顶点的度（这个顶点连接的边的数量），再定义一个u，这个u仅仅是一个与顶点的度有关的数。得到以下公式：(1 - n\*u) \* original\_position + u \* neighbor\_position\_sum。从这个式子可以看到，当顶点的度很大（连了好多三角形），那么这个顶点基本就可以由周围的老顶点决定位置，当顶点的度很小（连接的三角形很少），那么这个顶点由自己原来位置决定更新后位置的比重就会增大。



## 2、Catmull-Clark Subdivision (General Mesh)

刚刚的Loop Subdivision仅仅可以针对全部都是三角面的网格。而Catmull-Clark Subdivision适用于更普遍的情况，无论是四边面还是三角面都可以很好地进行细分处理。

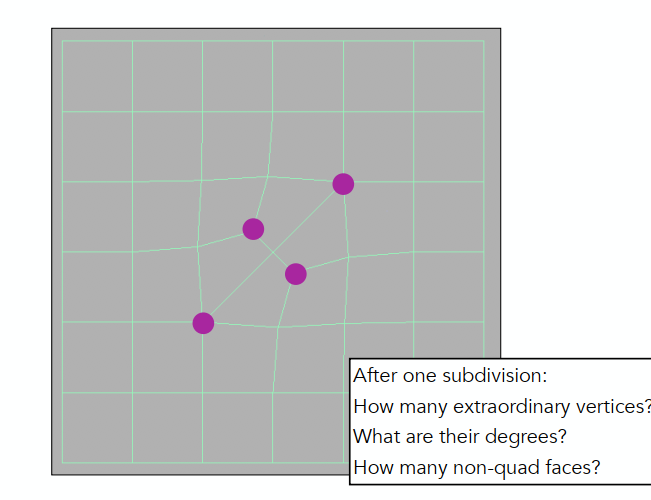


首先我们定义Non-quad face，所有不是四边面的网格形状都被称作非四边形面。

其次我们再定义奇异点（极点），所有度不为4的点都被称为奇异点(紫色点)

Each subdivision step:

1. Add vertex in each face找出每个非四边形面的边上的中点
2. Add midpoint on each edge找出每个非四边形面的中心
3. Connect all new vertices将中心与各个中点连接起来



如上图，即为经过Catmull-Clark细分后的样子。

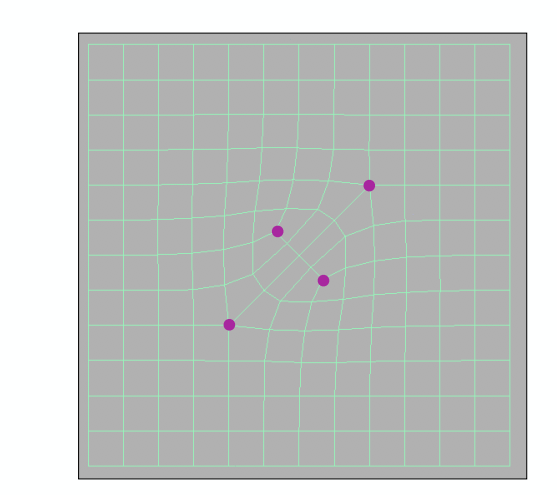
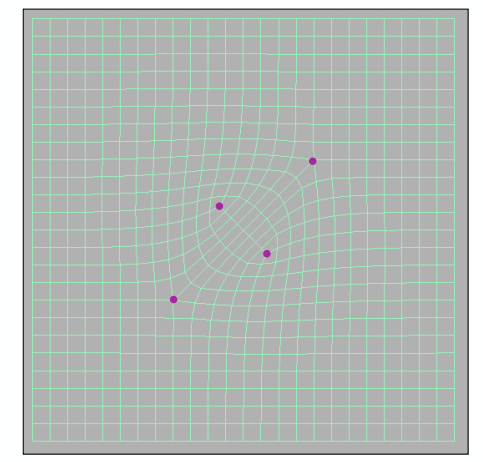
After one subdivision:

--How many extraordinary vertices? 共有4个奇异点

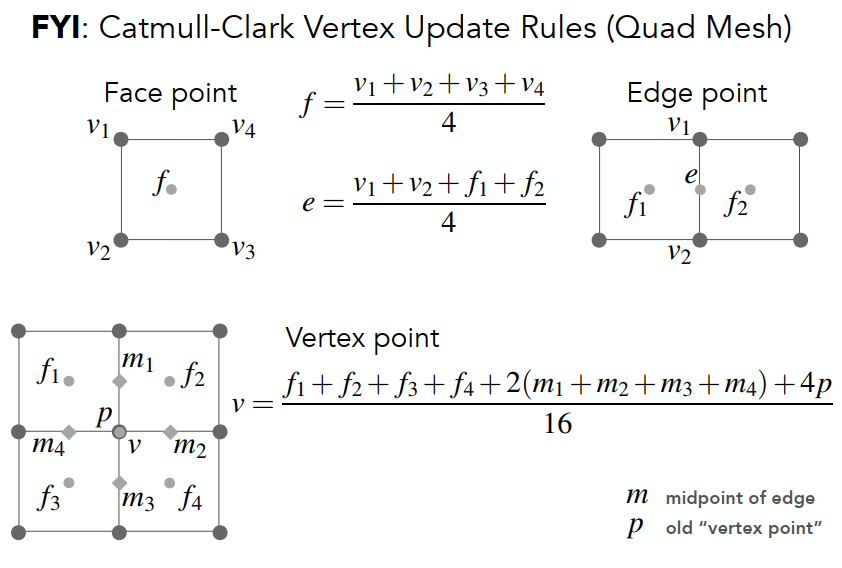
--What are their degrees? 原来是度是5的奇异点现在的度还是5，新引入的奇异点由于要与三角面上的每一条边的中点相连，因此度为3

--How many non-quad faces? 没有非四边形面了，全部都为四边形面（细分的过程中每一个非四边形面都会引入一个奇异点，然后非四边形面就消失了）

再次细分后，面会越来越光滑

FYI: Catmull-Clark Vertex Update Rules (Quad Mesh)



Catmull-Clark细分后对点的位置的更新分三种：

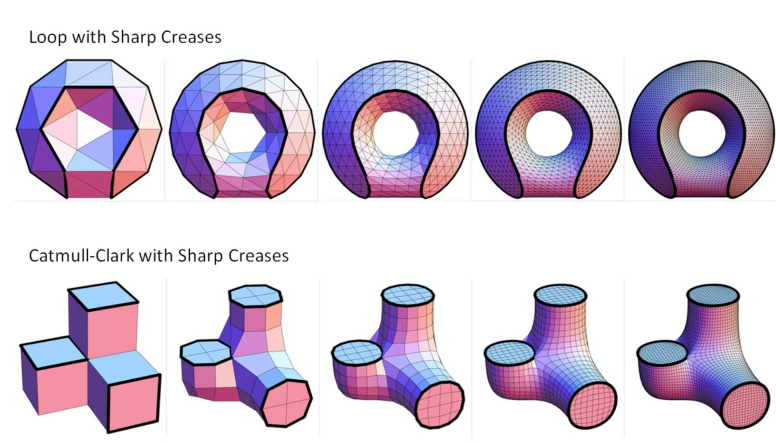
第一种是对于面的中心点f的更新

第二种是对于边的中心点e的更新

第三种是对于老的顶点v的更新。公式如上图所示

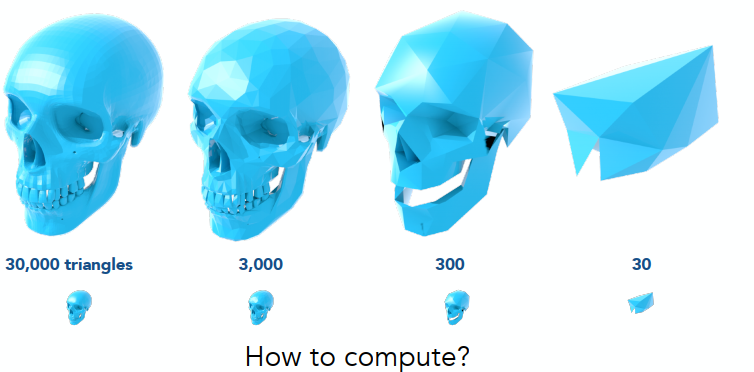
Convergence: Overall Shape and Creases

Loop Subdivision仅能用于三角形面的网格，Catmull-Clark Subdivision可以用于任何形状组成的网格。



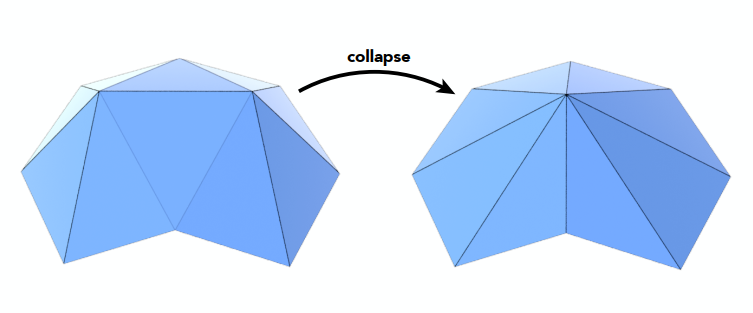
# (二)Mesh Simplification简化

Goal: reduce number of mesh elements while maintaining the overall shape



不同情况下要用不同面数的模型，那么如何去计算对网格的简化？

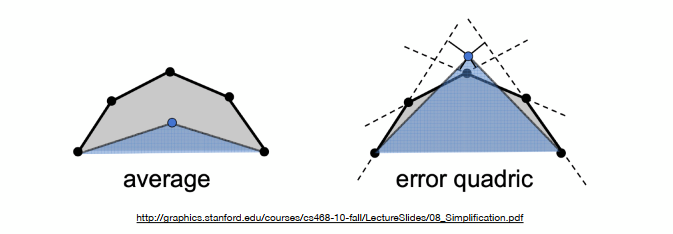
Collapsing An Edge(边坍缩)



边坍缩：形象的说明的话，即为一条边两端连着两个顶点，将这两个顶点往中间一捏，变成一个点，这个操作即为边坍缩。

这里提供边坍缩的方法。

## Quadric Error Metrics（⼆次误差度量）



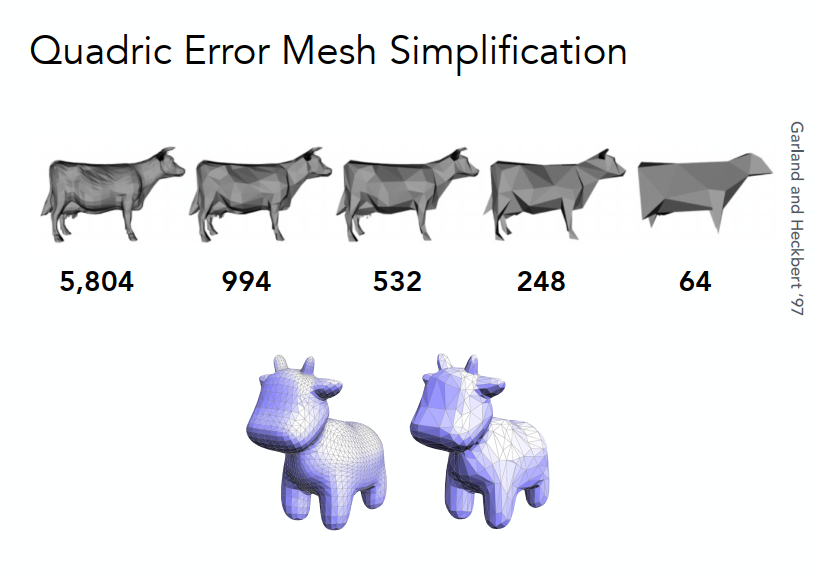
如上图左图，假设要简化上面三个顶点，变成一个顶点，该把这个顶点放在哪里，才能保证简化后的蓝色三角形与原本灰色多边形基本保持轮廓的一致？这里引入二次误差度量概念：我们希望把这个顶点放某个位置上，可以最小化二次误差。在此处的意思就是：把这个顶点放置在某个位置上，使得这个顶点到各个面（如上图右图，在这里看上去像是边）的距离的平方和达到最小。

对于整个模型有很多条边，假设如果坍缩一条边，并且算出把这个坍缩后的点放在最佳的位置上会得到一个多大的二次度量误差。对于整个模型而言，每次肯定都是要坍缩二次度量误差最小的边。

先给每个边打上一个分数，这个分数就是这个边的二次度量误差，从小的开始一个个进行坍缩。

但是问题是，如果坍缩了一条边，有一些其他的边的位置要跟着这条坍缩的边变化，那么这些边的二次度量误差就会随之发生变化。

因此这里我们要通过一种数据结构，既可以每次取到二次度量误差的最小值的边，又可以动态地以最小的代价去更新其他的受影响的元素。那么这里要使用的数据结构就是堆（优先队列）。



# (三)Shadow Mapping

An Image-space Algorithm图像空间算法

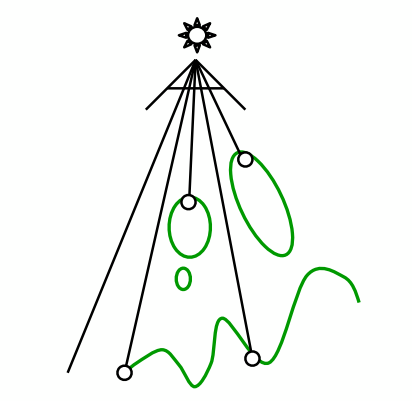
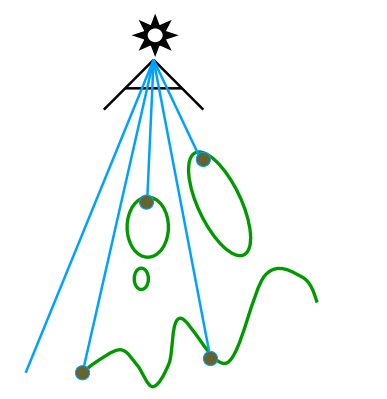
​ --no knowledge of scene’s geometry during shadow computation在阴影计算的过程中不需要了解场景的几何形状

​ --must deal with aliasing artifacts大部分阴影映射都有走样现象

Shadow Mapping最重要的一个思想是如果有个点不在阴影里，但又能被你看到，那么说明你可以从摄像机看到这个点，并且光源也可以看到这个点。

## 1、Step

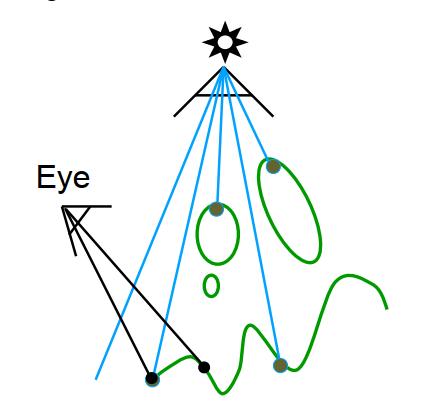
### Pass 1: Render from Light

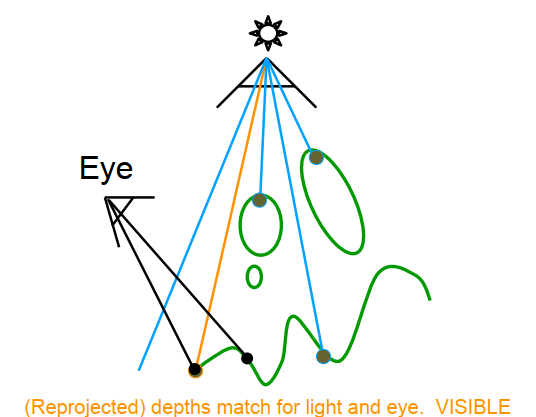
首先，从光源看向场景（可以假设光源有一个虚拟的摄像机看向场景）（如上图 ），绿色的圆圈就挡住了部分视线，然后得到一张光源处可以看到的图，这张图我们不需要着色，只需要把深度记录下来即可，得到一张从光源看向场景的深度图。

### Pass 2A: Render from Eye

然后从摄像机（眼睛）再向场景看。

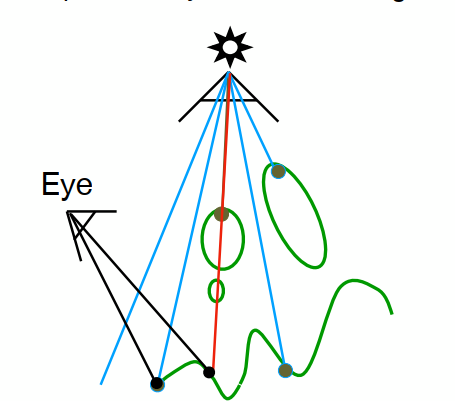


Pass 2B: Project to light



当从摄像机（眼睛）看到一个点时，我们可以将其投影回刚刚光源处虚拟相机成像平面上（点与光源连线，如上图橙色线），得到光源到这个点的深度信息。

Project visible points in eye view back to light source

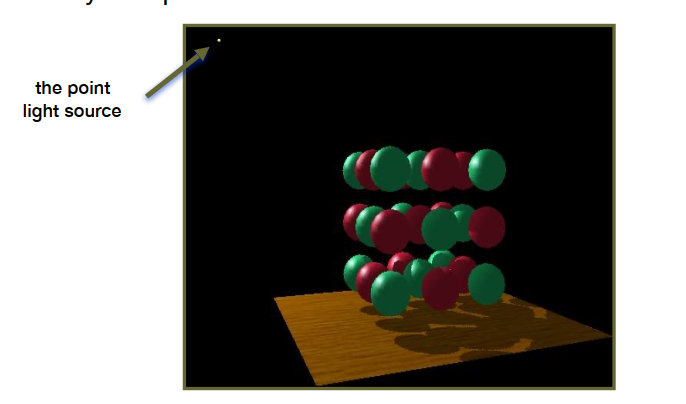


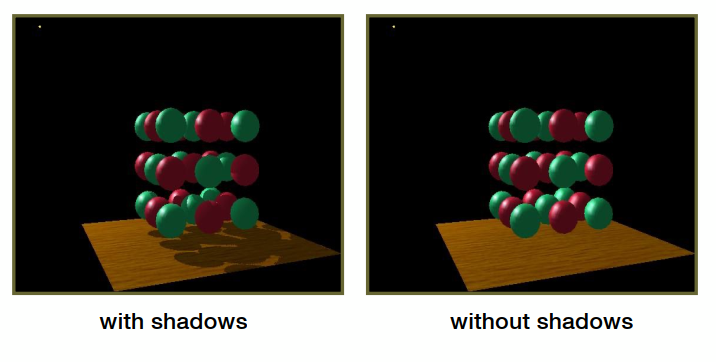
这一步同时可以得到光源到这个点所在方向上的深度信息原来记录在虚拟相机生成的深度图上的像素位置，从而得到这个像素上原来从光源看向场景（Pass 1中）的深度信息。

比较二者的深度信息，如果是一致的，这说明这个点完全可以被光源所看到。

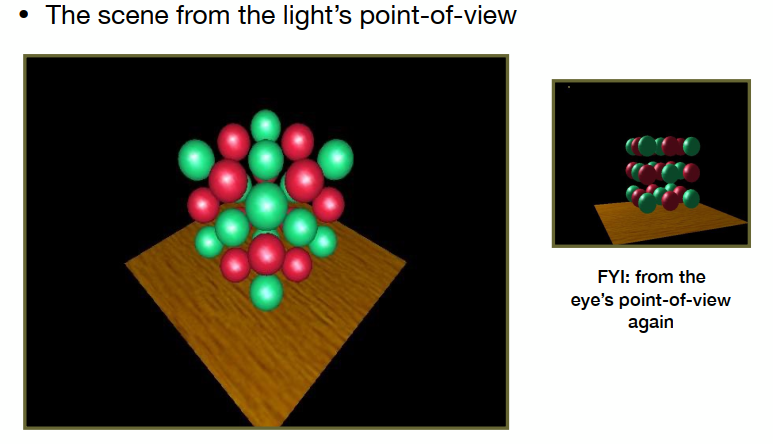
如果是这样一个点，同样我们可以投影回光源所在的虚拟相机所成像的图像上，我们找到那个像素，但是我们之前在那个像素上记录的深度是到绿色圆圈的，眼睛看向这个点投影回光源所在的虚拟相机的实际深度与从光源看向这个方向的最大深度不一致，这就说明这个点一定是之前从光源往这个方向看不到的点。因此这个点在阴影中。

## 2、Visualizing Shadow Mapping



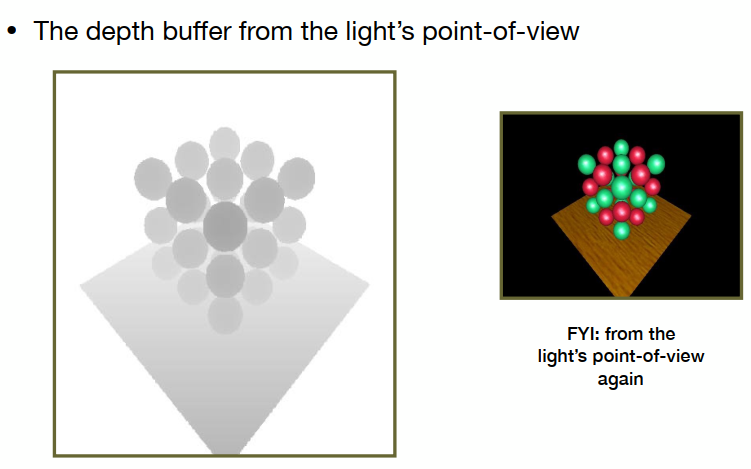


1. The scene from the light’s point-of-view



The depth buffer from the light’s point-of-view

首先从光源看向场景，得到深度图

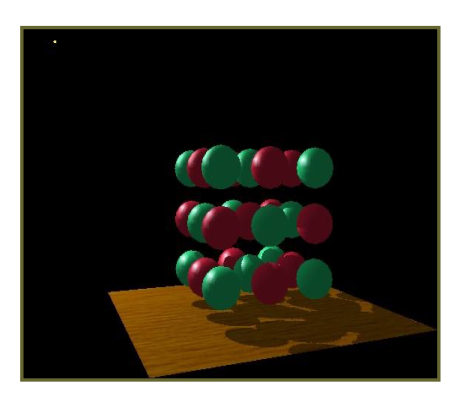


1. 、Comparing Dist(light, shading point) with shadow map



然后从摄像机看向场景，投影回光源也得到一张深度图

两张深度图相同方向的信息比较，得到阴影区域 。

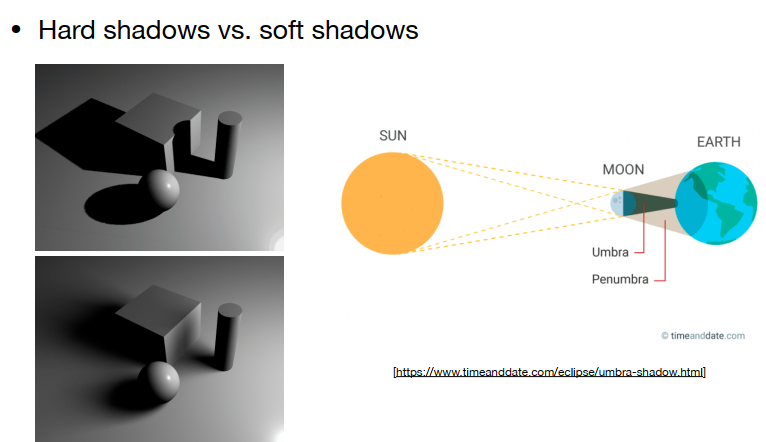


## 3、Problems with shadow maps

这种方式会导致阴影的硬边缘、分辨率、数值精度等问题。

另一种方式叫软阴影

### Hard shadows vs. soft shadows



效果如上图所示。软阴影会有过度，越靠近物体根部位置会越硬，离得越远就会越虚化（越软）。

其效果的原理其实是一种自然现象，软阴影的部分其实是物理中的半影。如果一个地方完全看不到光源，这个地方就被叫作本影，如果能部分看到光源就被称为半影。这里我们不能把光源简单看成一个点，而是有体积的。