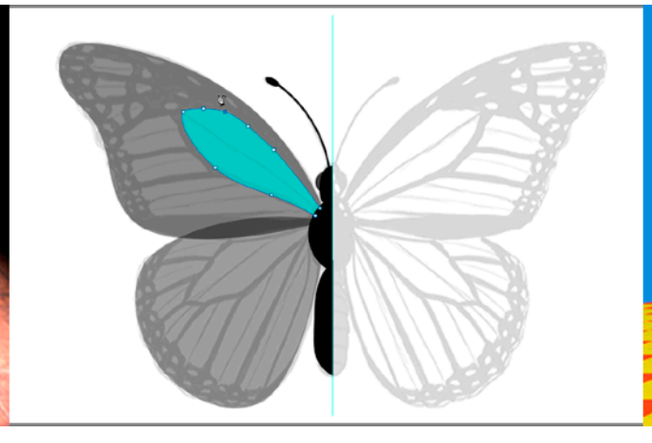
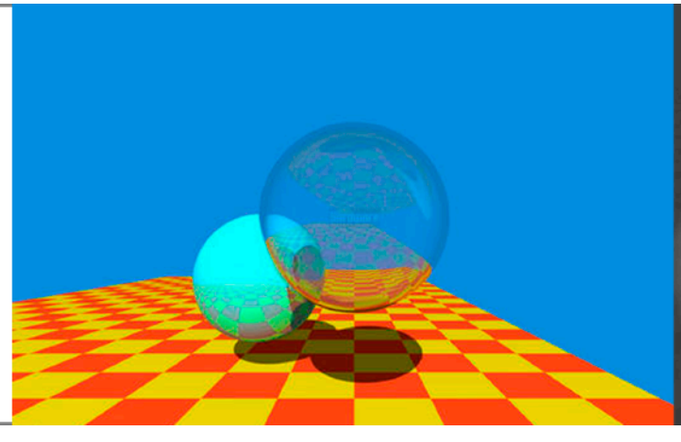
Introduction to Computer Graphics

GAMES101, Lingqi Yan, UC Santa Barbara

Lecture 12:
Geometry 3


http://www.cs.ucsb.edu/~lingqi/teaching/games101.html

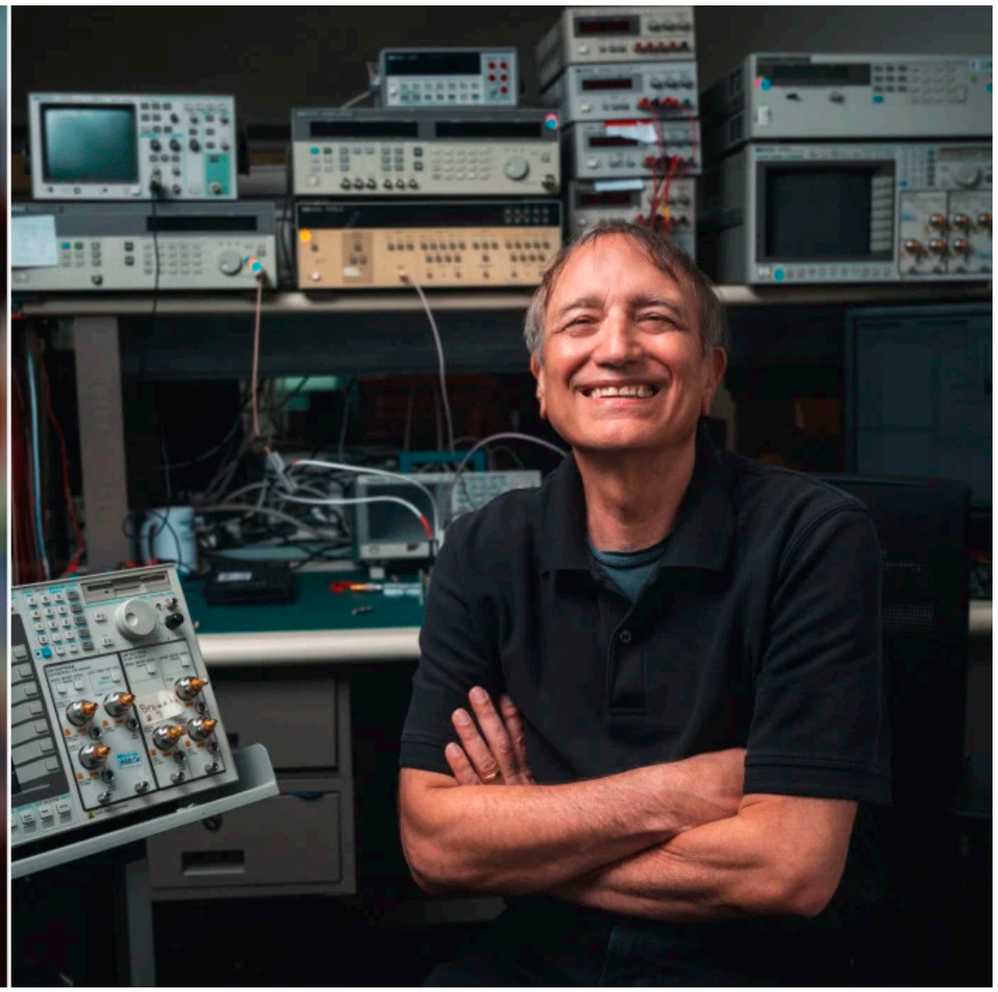
Announcements

- Homeworks
  - Enjoying HW3?
  - HW1 submission window reopened (similar policy applies to later HWs)
- The T/N/B calculation
  - Will be in the next lectures [local shading frame]
- BIG NEWS!
  - Computer Graphics won the Turing Award after 32 years!

Turing Award Winners

- Made Computer Graphics great
- We will soon learn about their work!

Ed Catmull

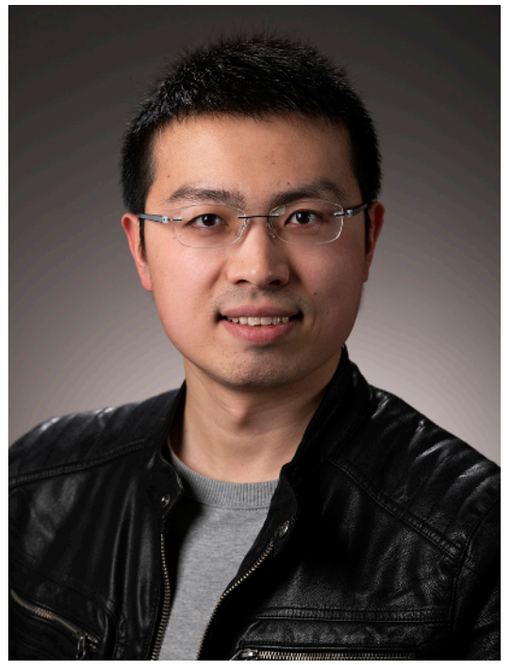


Pat Hanrahan

Academic Family Tree

Pat Hanrahan @ Stanford

Ravi Ramamoorthi @ UCSD



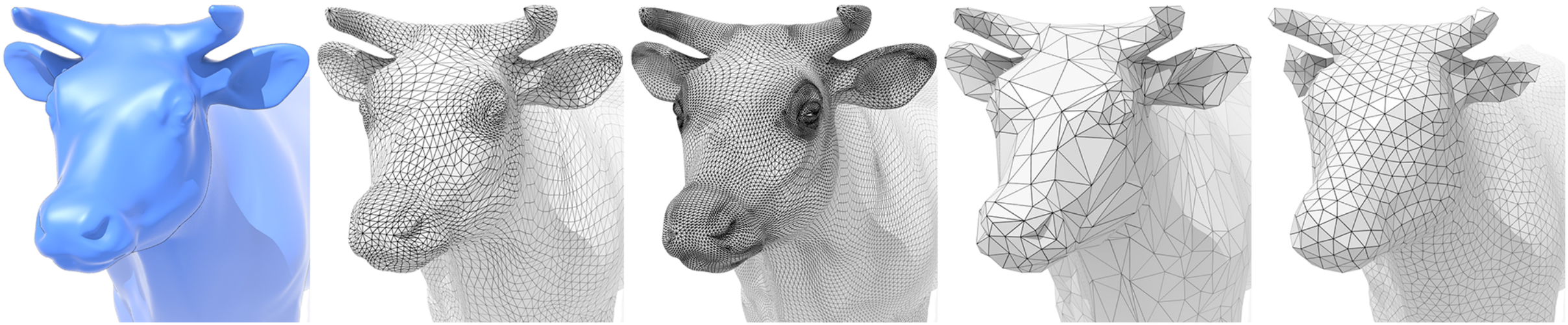
Lingqi Yan **@ UCSB**

Pradeep Sen **@ UCSB**

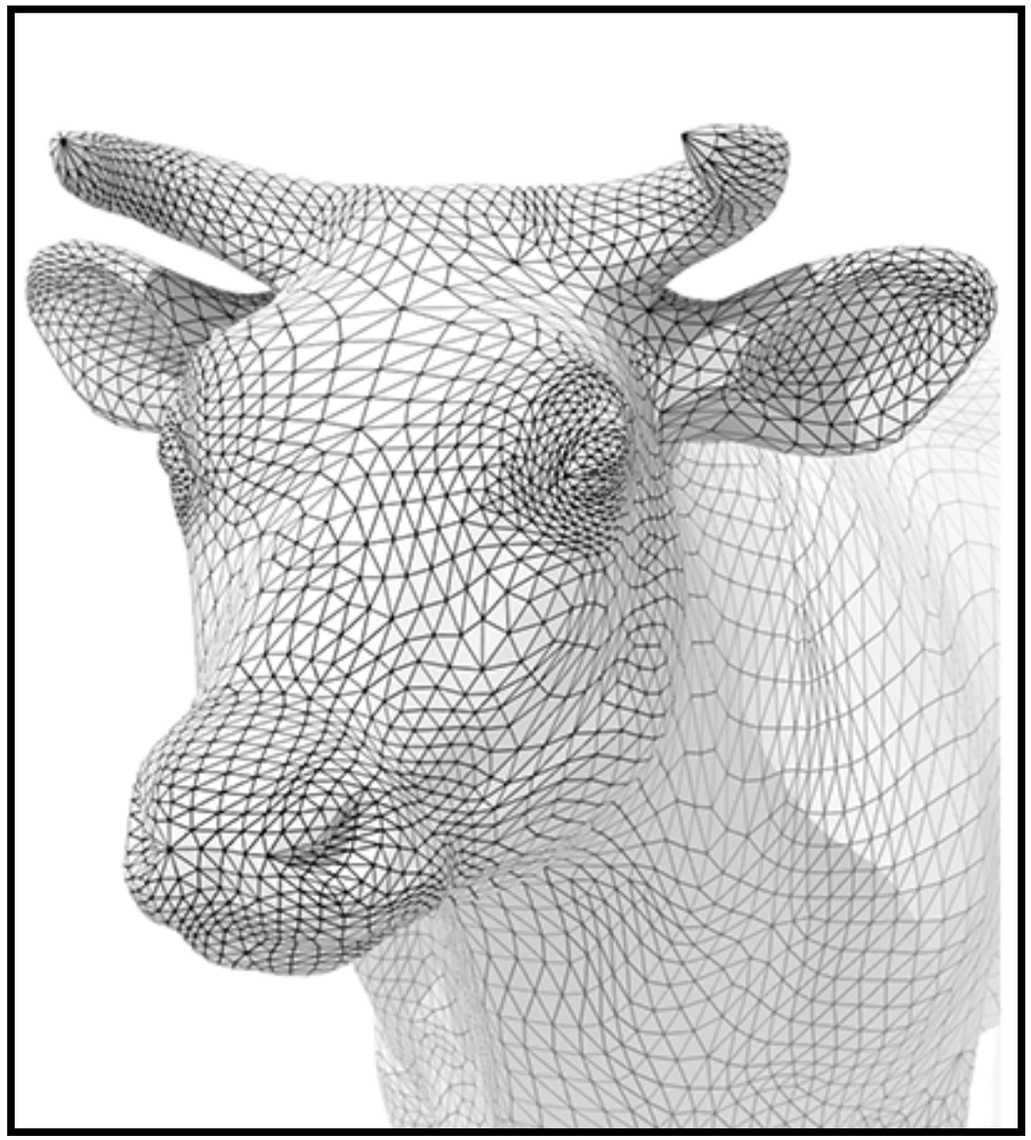
Back to Geometry

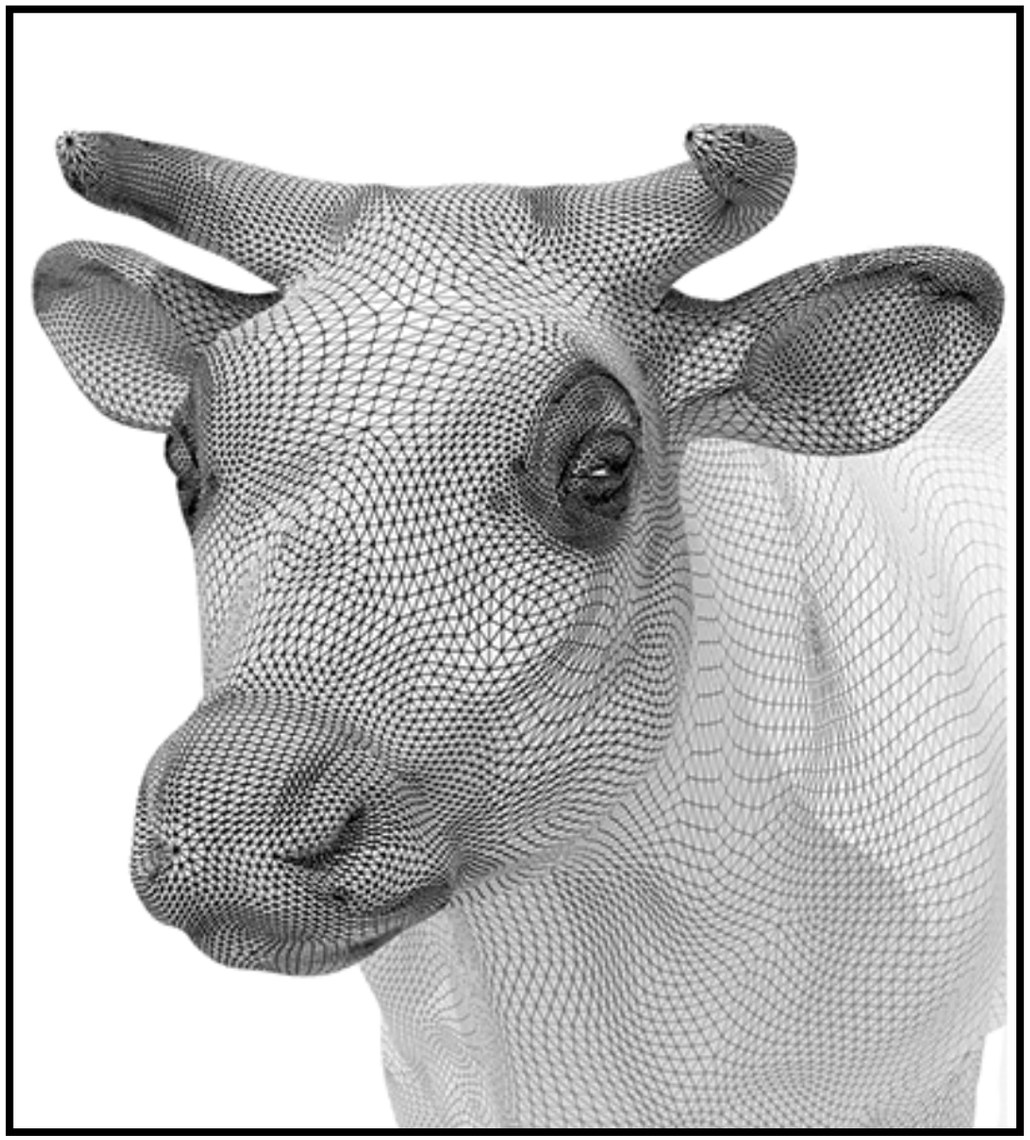
Mesh Operations: Geometry Processing

- Mesh subdivision
- Mesh simplification
- Mesh regularization



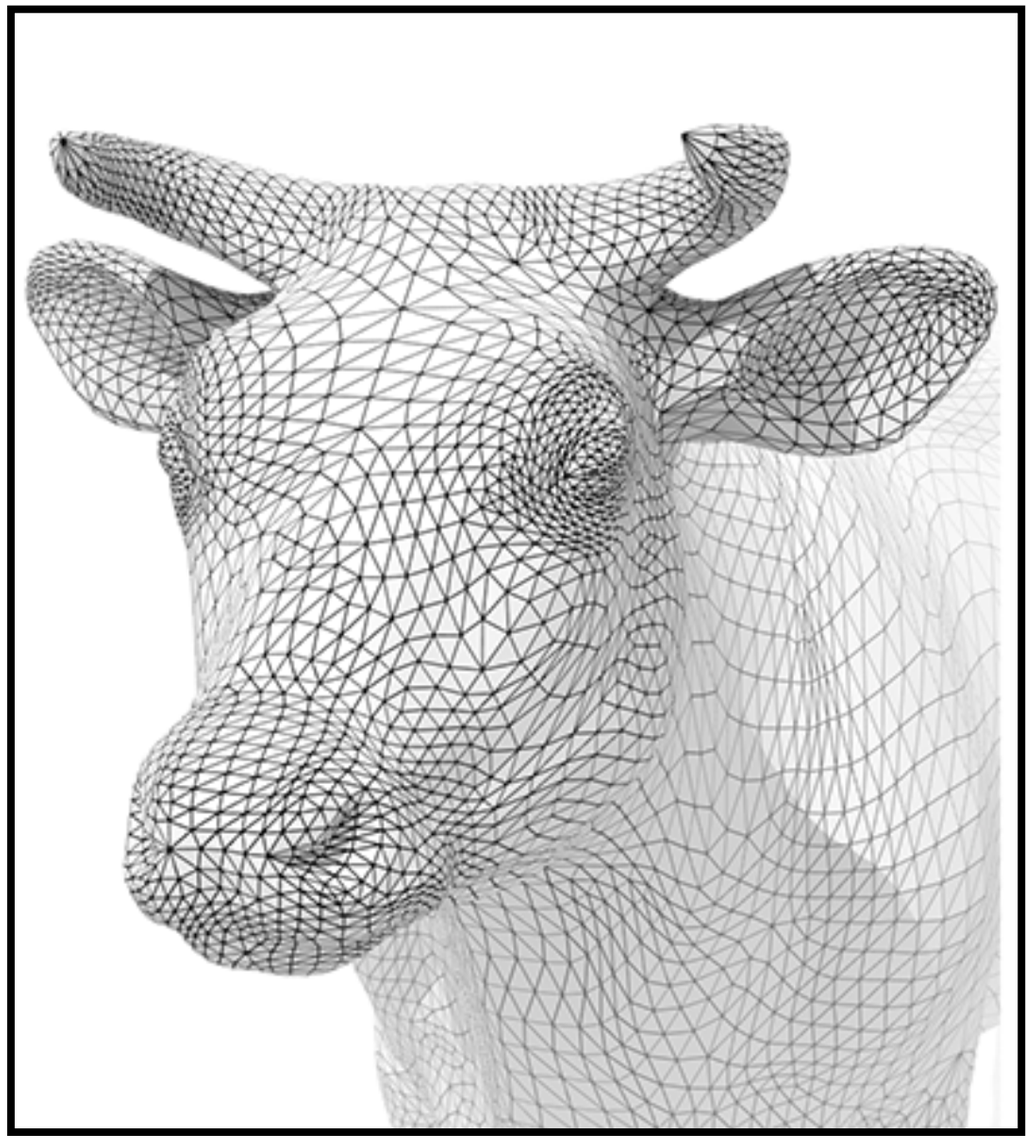
Mesh Subdivision (upsampling)

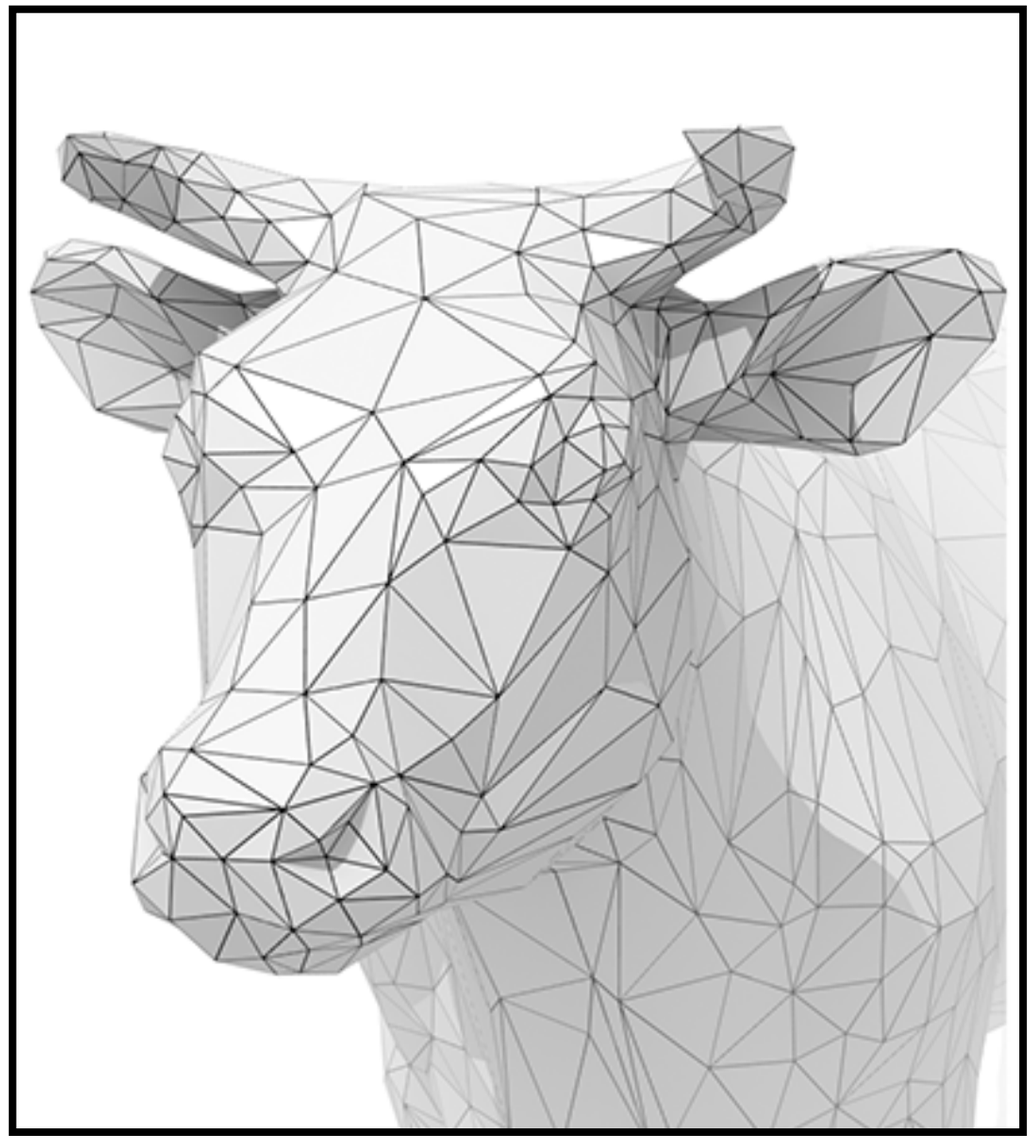




Increase resolution

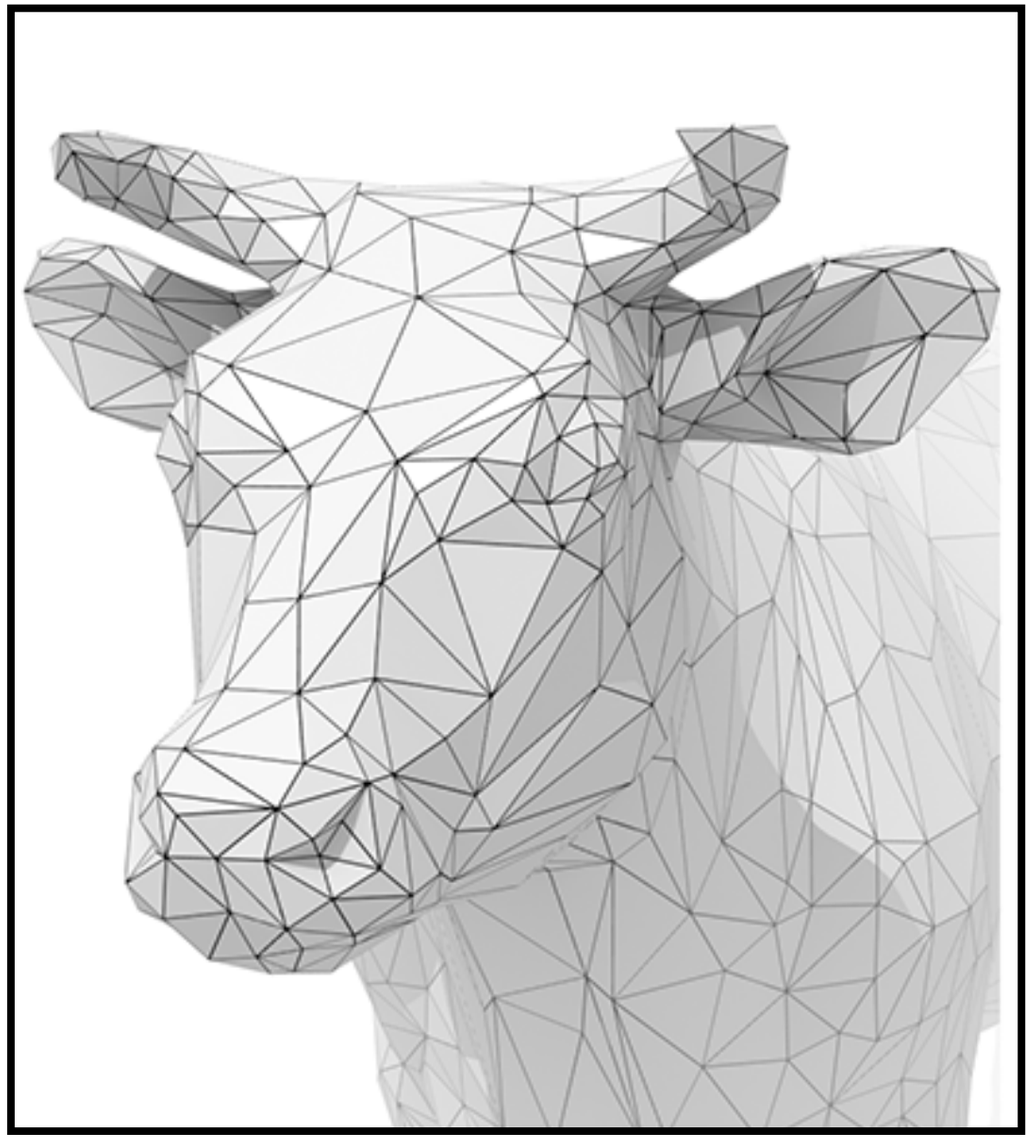
Mesh Simplification (downsampling)

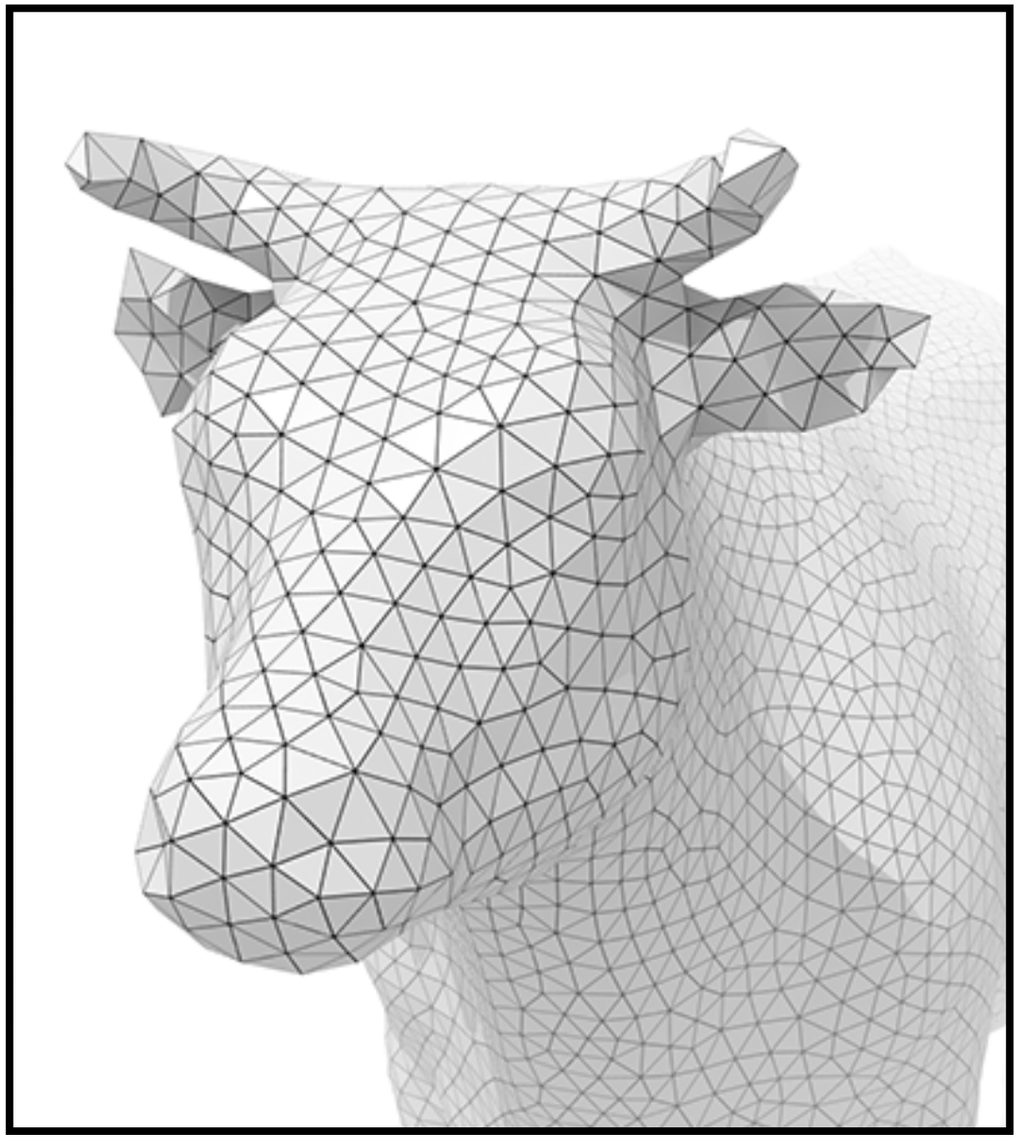




Decrease resolution; try to preserve shape/appearance

Mesh Regularization (same #triangles)





Modify sample distribution to **improve quality**

Subdivision

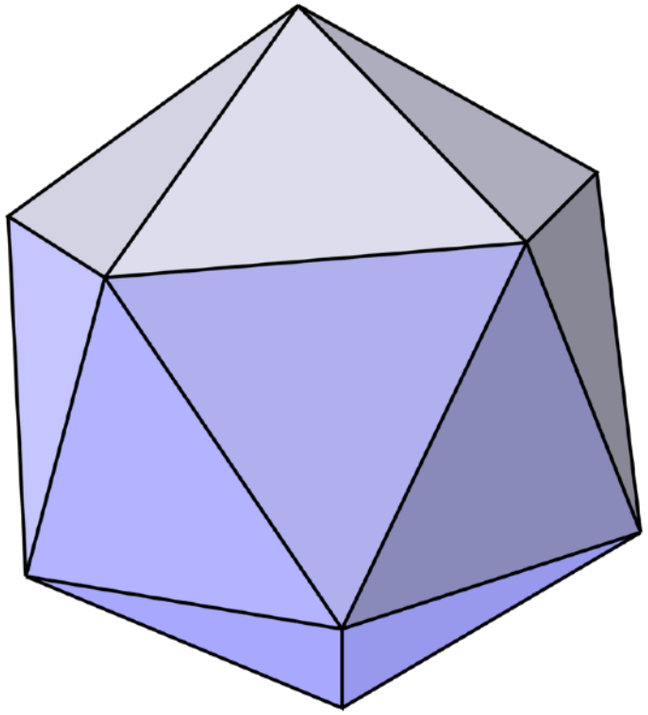
Loop Subdivision

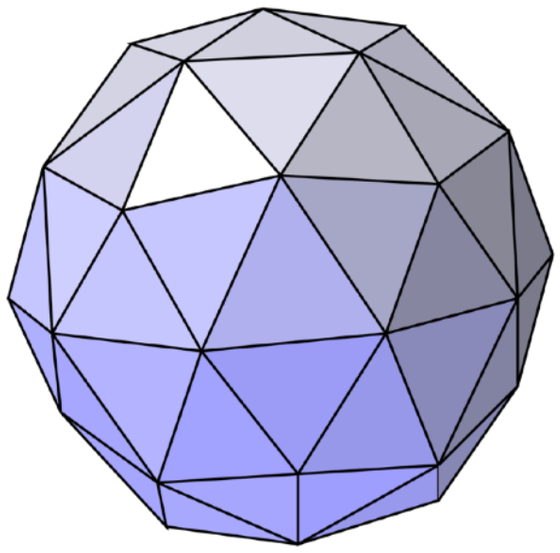
LOOP(名称，不是循环的意思)细分: 连接各边中点，并重新改变各个顶点位置，从而创造出更多三角形面，使得表面更加光滑

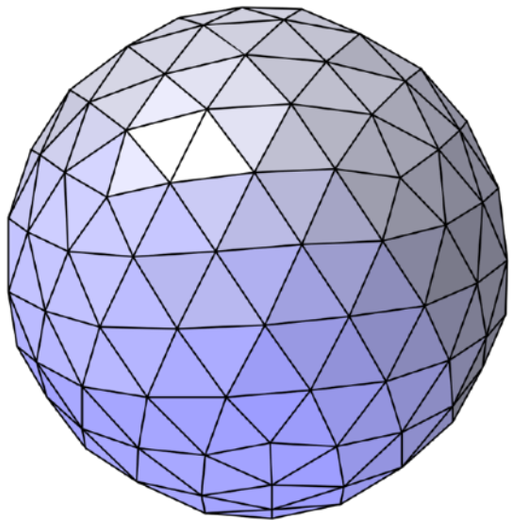
Common subdivision rule for triangle meshes

First, create more triangles (vertices)

Second, tune their positions







Simon Fuhrman

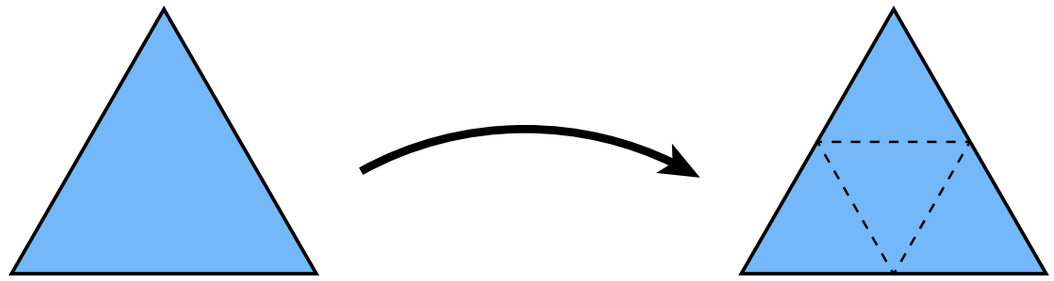
Loop Subdivision

简单而言分为两步：

1. 创建更多的三角形
2. 调整三角形的顶点位置

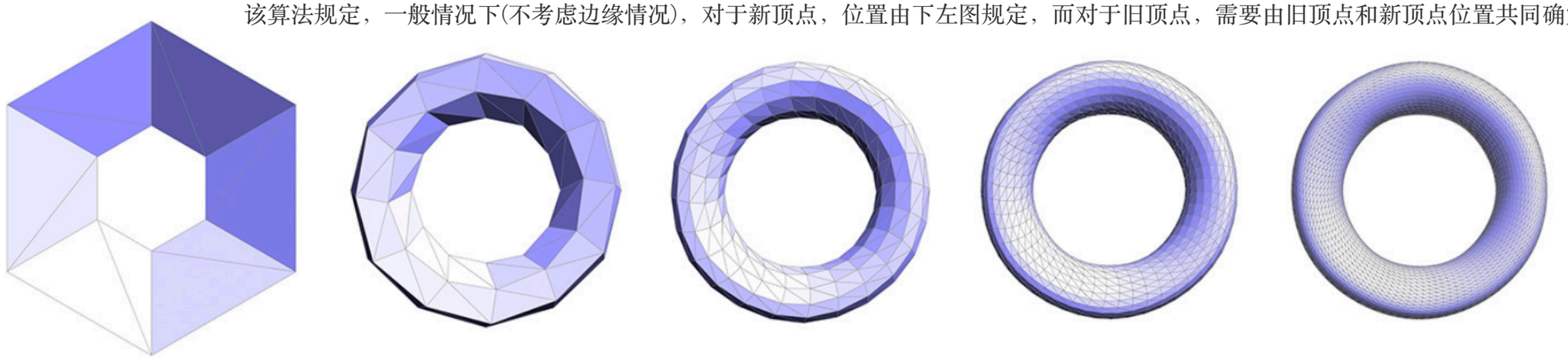
- Split each triangle into four

首先基于每个三角形的边中点，将任一三角形细分为四个。
但仅仅细分三角形是不会在视觉上产生差异的。
需要再调整各个顶点的位置。
对新的顶点和老的顶点的位置更新算法略有不同。



- Assign new vertex positions according to weights
  - New / old vertices updated differently

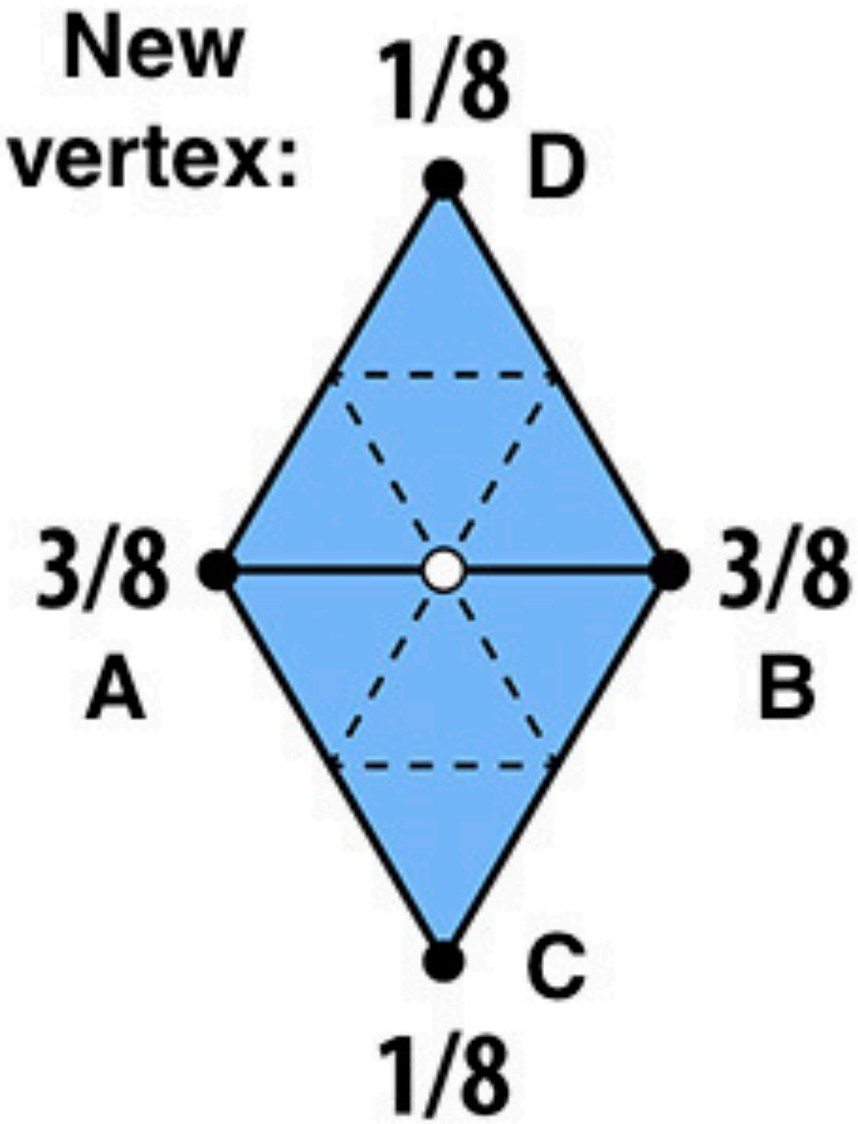
该算法规定，一般情况下(不考虑边缘情况)，对于新顶点，位置由下左图规定，而对于旧顶点，需要由旧顶点和新顶点位置共同确定



Loop Subdivision — Update

新顶点更新算法

For new vertices:



Update to:

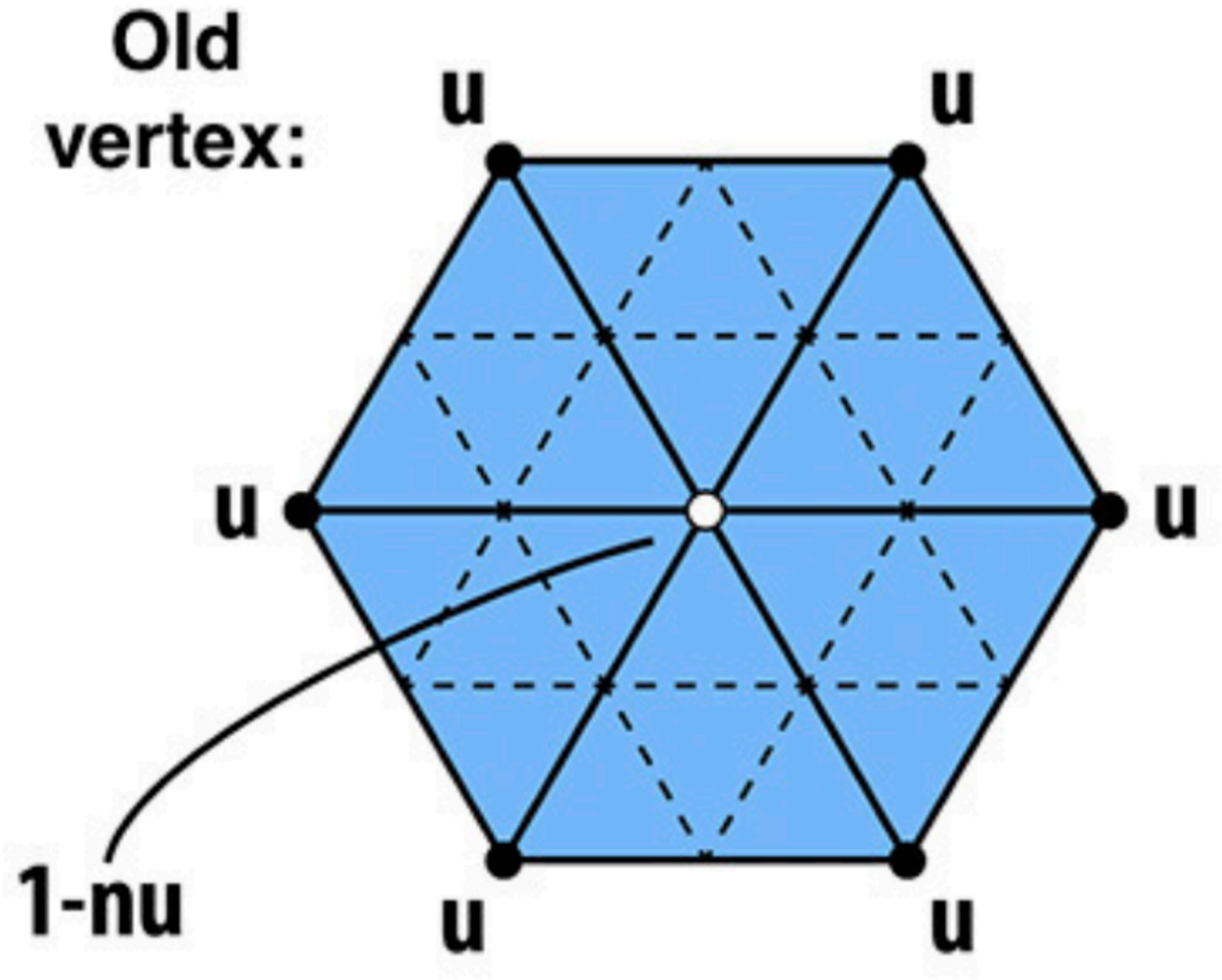
3/8 * (A + B) + 1/8 * (C + D)

Loop Subdivision — Update

旧顶点的更新算法

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

核心思路就是基于自己和与自己相临的其他老顶点的高度进行加权平均



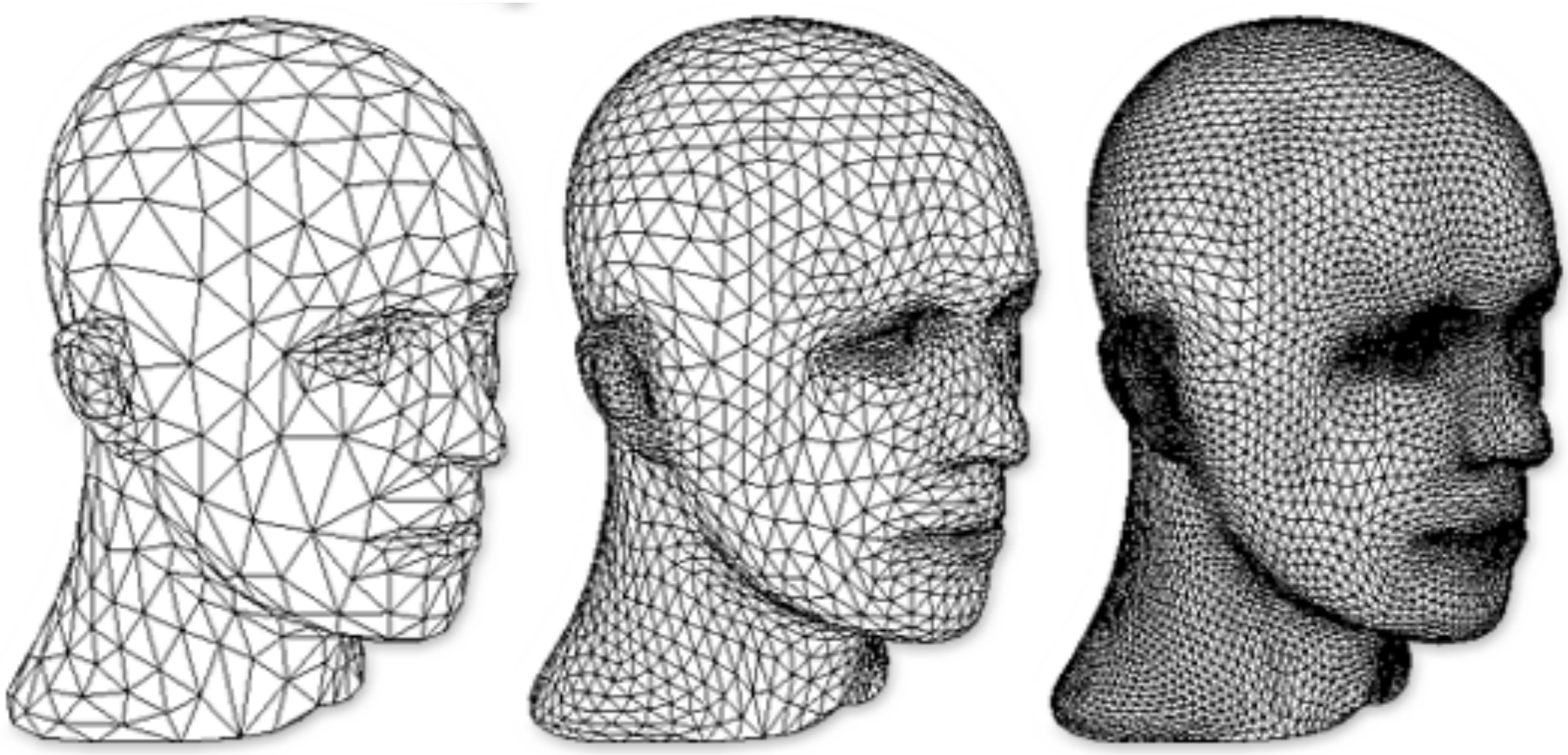
Update to:
(1 - n*u) * original_position +
u * neighbor_position_sum

n为该顶点的度(依附于某个顶点的边的条数)，u为一个和n有关的数

n: vertex degree

u: 3/16 if n=3, 3/(8n) otherwise

Loop Subdivision Results

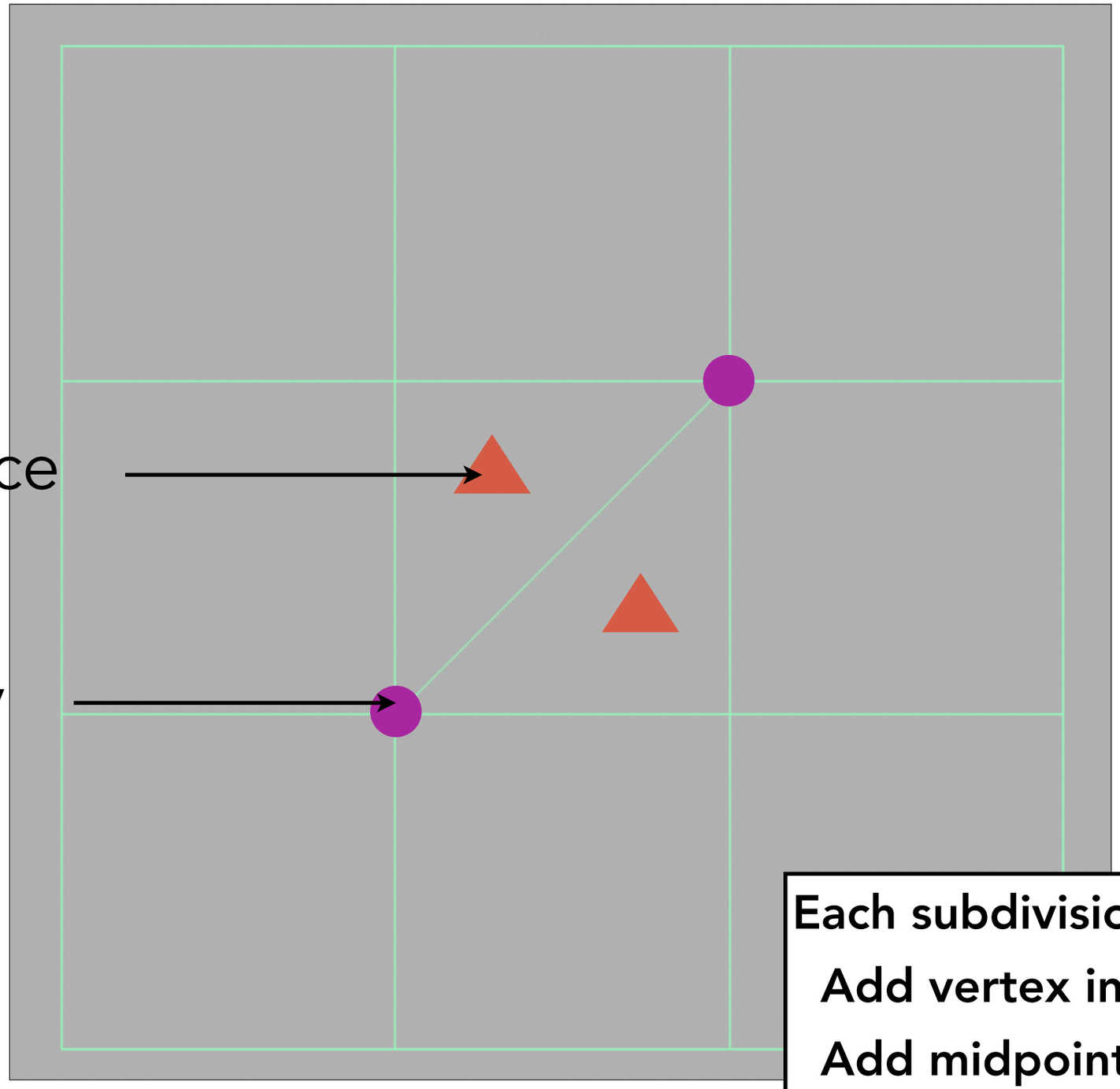


Catmull-Clark Subdivision (General Mesh)

loop细分有一个前提，即只适用于三角形网格，而对于非三角形网格的细分，就需要借助catmull-clark算法 该算法定义面分为两种——四边面和非四边面，并定义度为4的顶点为非奇异点，其余点均为奇异点

Non-quad face

Extraordinary vertex (奇异点) (degree != 4)

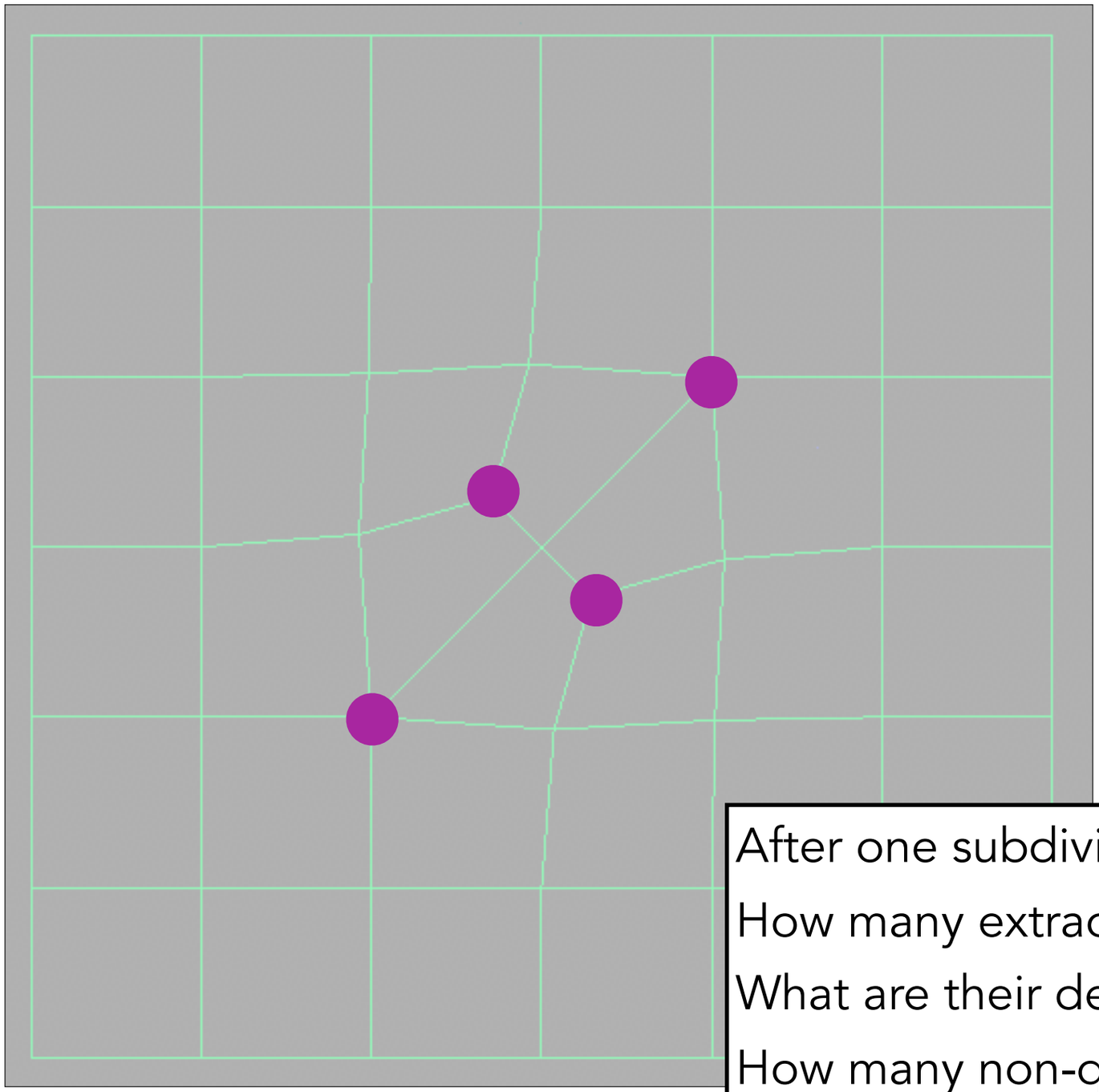


其主要针对于不全是由三角形构成的 Mesh。
考虑四边形为正常的几何结构。四边形细分非常容易，取中点连线再加权即可。

Each subdivision step:

- **Add vertex in each face**
- **Add midpoint on each edge**
- **Connect all new vertices**

Catmull-Clark Subdivision (General Mesh)



具体做法是，对每个非四边面都取其中的一个点 (重心或者其他点)，将其与该面的其他顶点分别连接，在这个过程中，会引入一个新的奇异点，并且在一次细分后，所有非四边面都变为了四边面，在后续的细分中，将不会引入新的奇异点

After one subdivision:

How many extraordinary vertices?

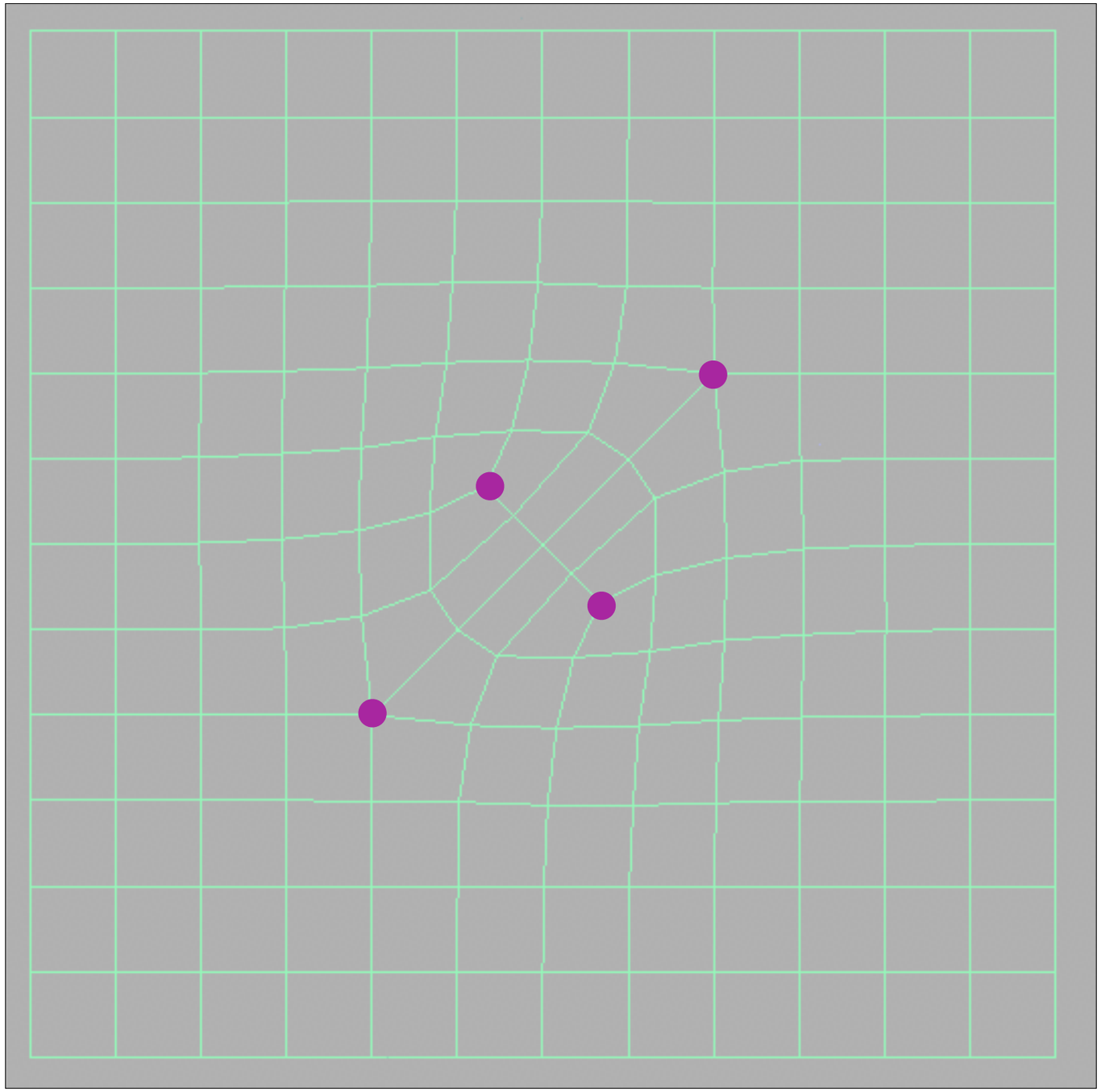
What are their degrees?

How many non-quad faces?

Catmull-Clark Subdivision (General Mesh)

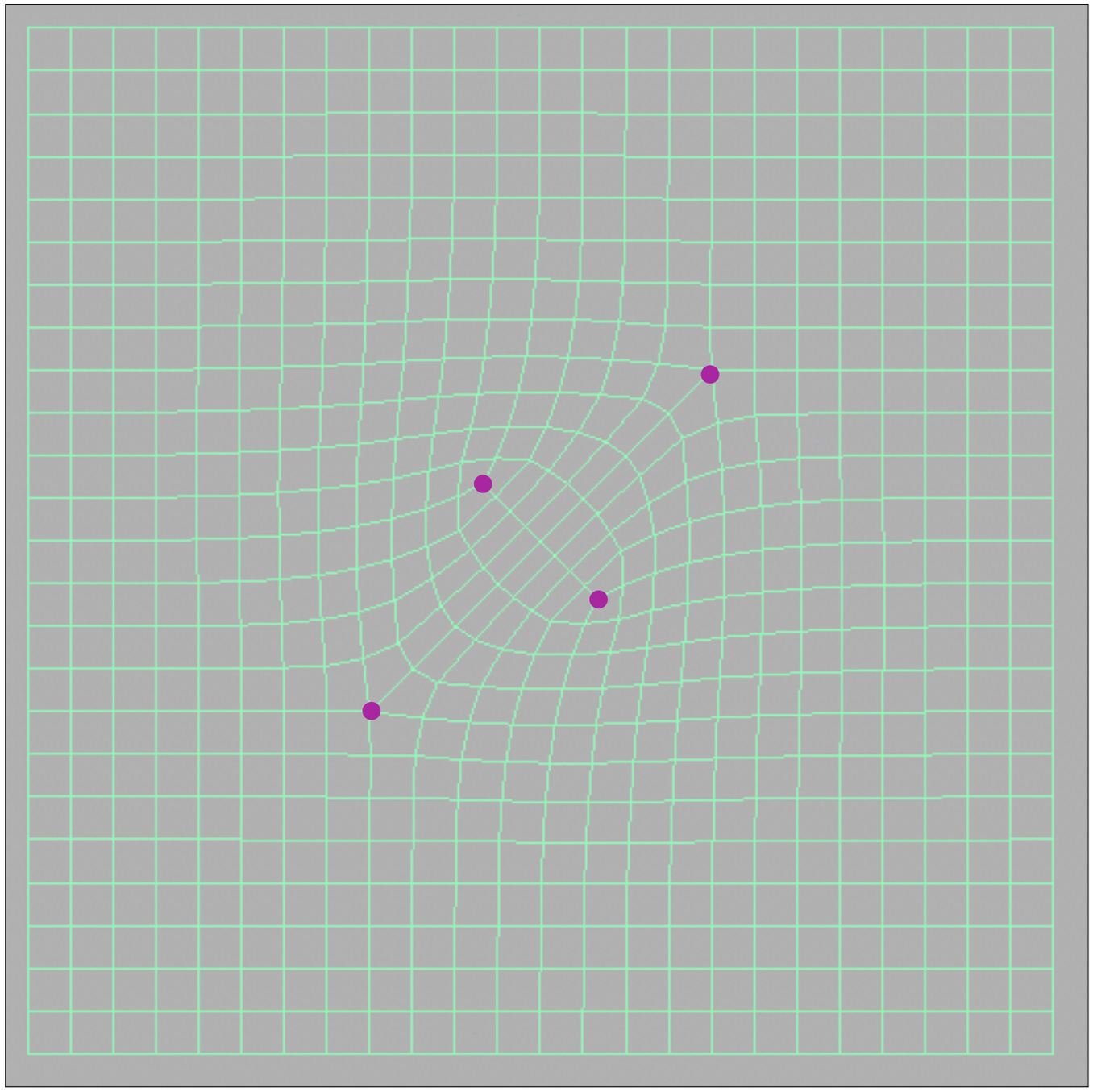
对于细分后顶点位置的调整，先将顶点分为三大类

1新的在面上的点;2新的在边上的点;3旧的点



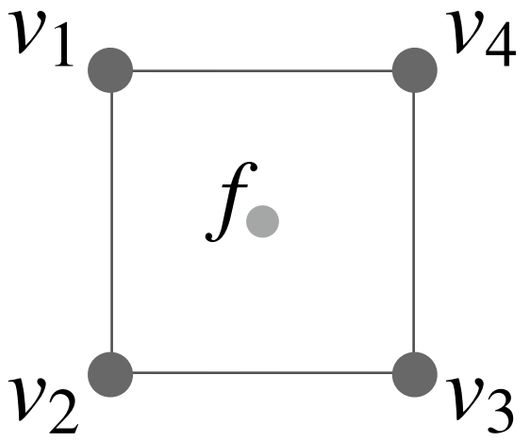
且可以发现，细分之后全部都是四边形，不再存在三角形。但奇异点会继续保持为奇异点

Catmull-Clark Subdivision (General Mesh)



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

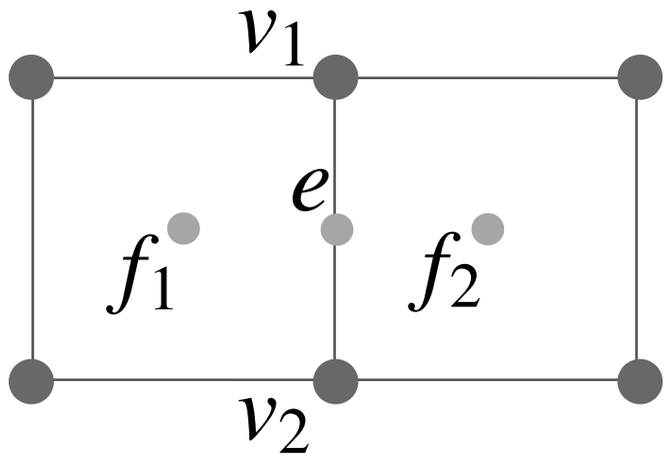
Face point

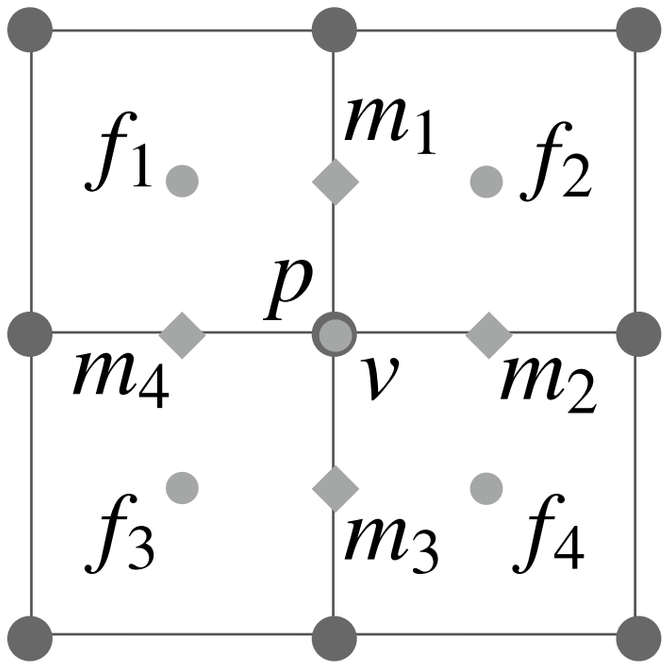


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$

Edge point





Vertex point

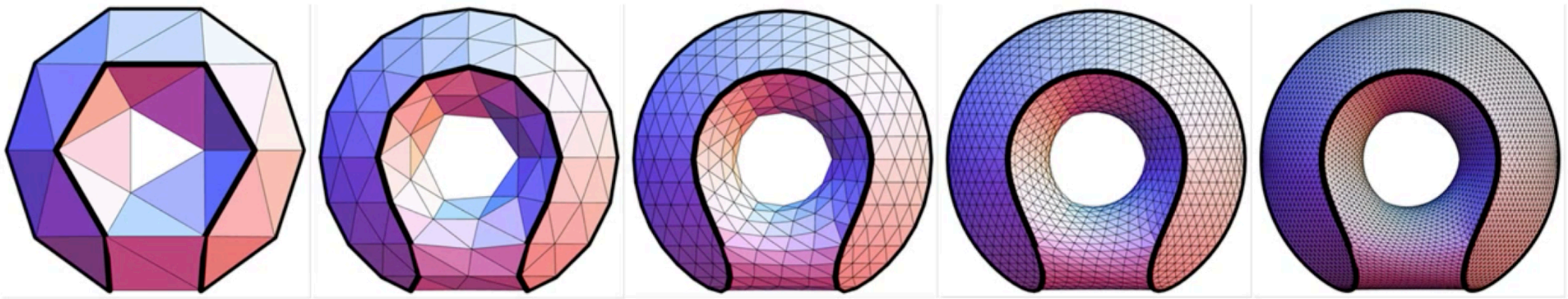
$$v = \frac{f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p}{16}$$

m midpoint of edge

p old “vertex point”

Convergence: Overall Shape and Creases

Loop with Sharp Creases



Catmull-Clark with Sharp Creases

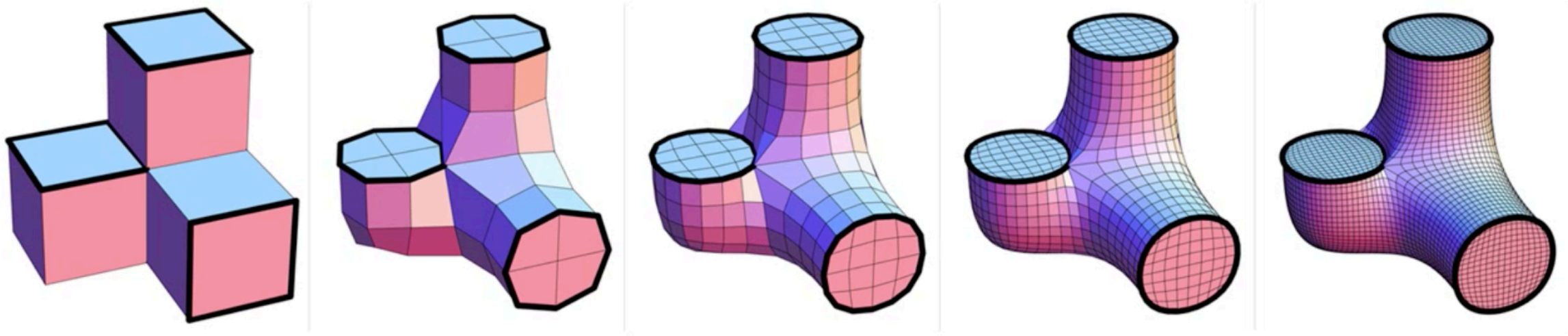


Figure from: Hakenberg et al. Volume Enclosed by Subdivision Surfaces with Sharp Creases

Subdivision in Action (Pixar's "Geri's Game")

Mesh Simplification

网格简化

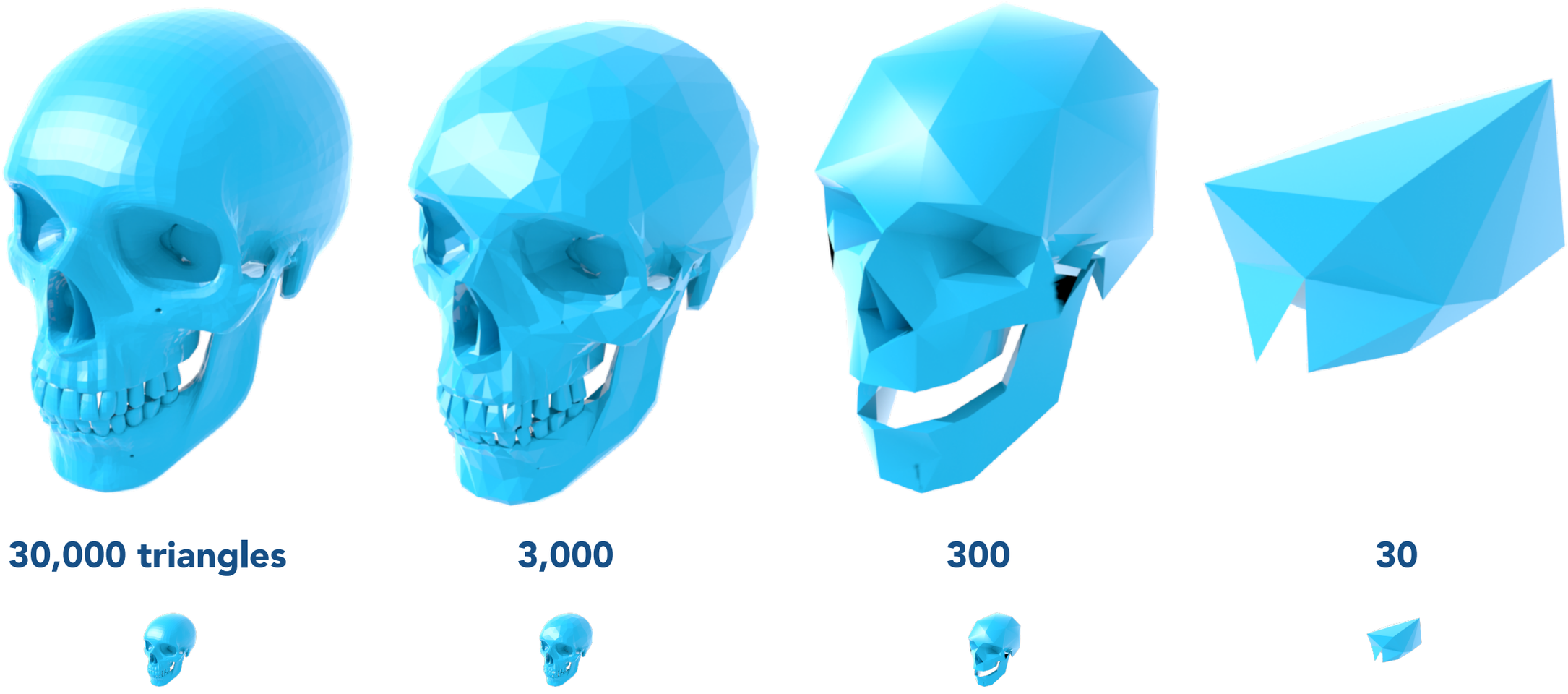
区别于细分，Mesh 简化也是常见的需求。

特别对于游戏，实时渲染需要控制场景的总面数，一种合理的策略是让较远的物体面数偏低，近处的物体面数偏高。

为进一步提升实时性，通常需要同时保存多级模型，即 LOD。分级的过程能自动化则远比手动调整效率高。

Mesh Simplification

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

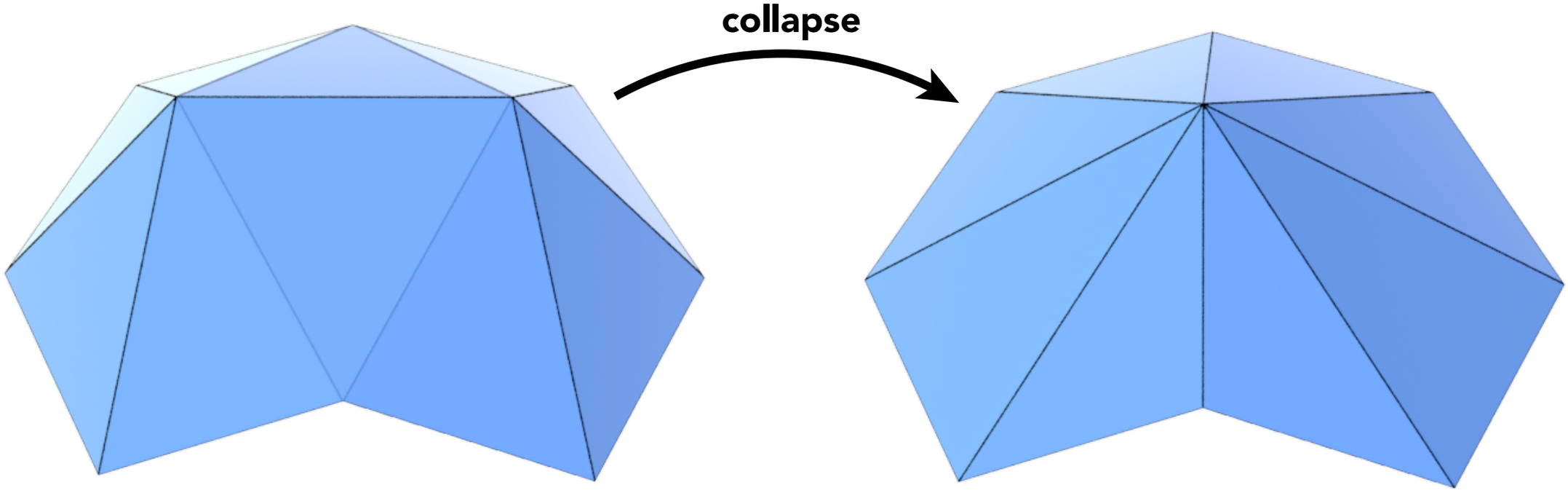


30,000 triangles | **3,000** | **300** | **30**

How to compute?

Collapsing An Edge

- Suppose we simplify a mesh using edge collapsing



为达到这个目的，边坍缩是一种容易想到的方式。即把相邻的两个点压缩到一起

但实际操作中这个方法并不容易，因为其面临一些隐藏问题：

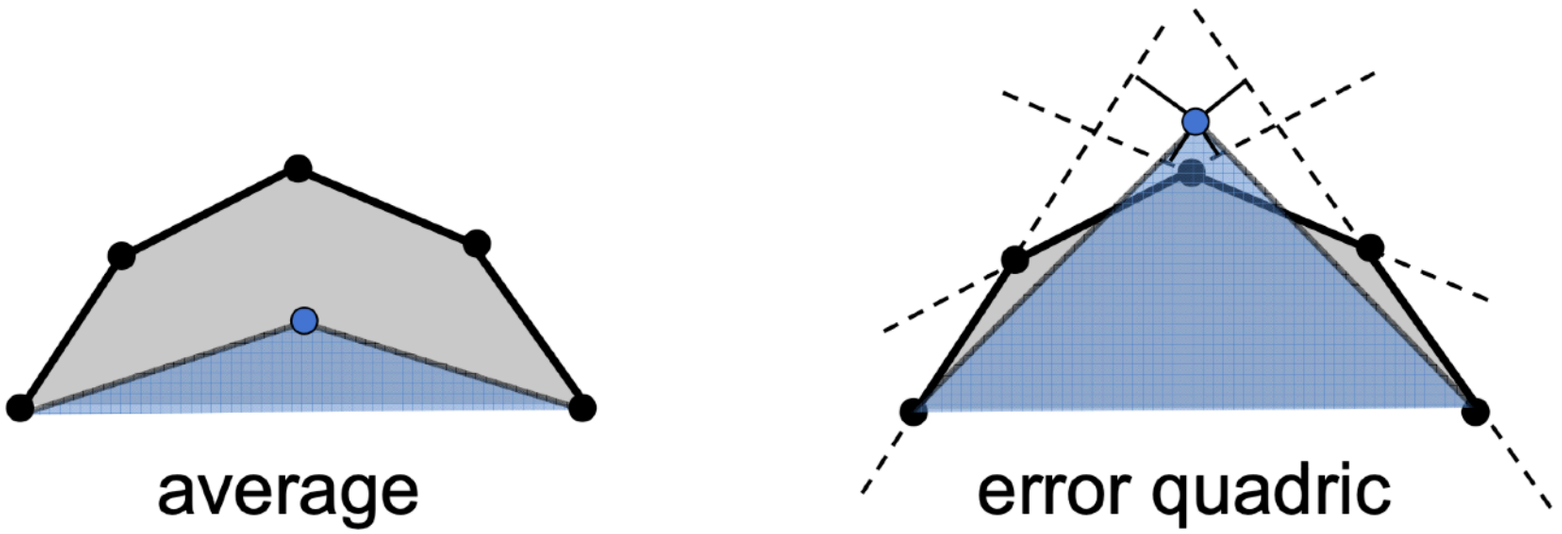
1. 坍缩哪些边？理应优先坍缩不重要的边，但哪些边是不重要的呢？
2. 坍缩后的顶点位置也不应该是平均值位置，如图而言讲道理需要高一些，但如何描述呢？

Quadric Error Metrics

（二次误差度量）

度量贡献度和寻找最小误差的一个方法, 核心思路非常类似于线性回归的误差度量

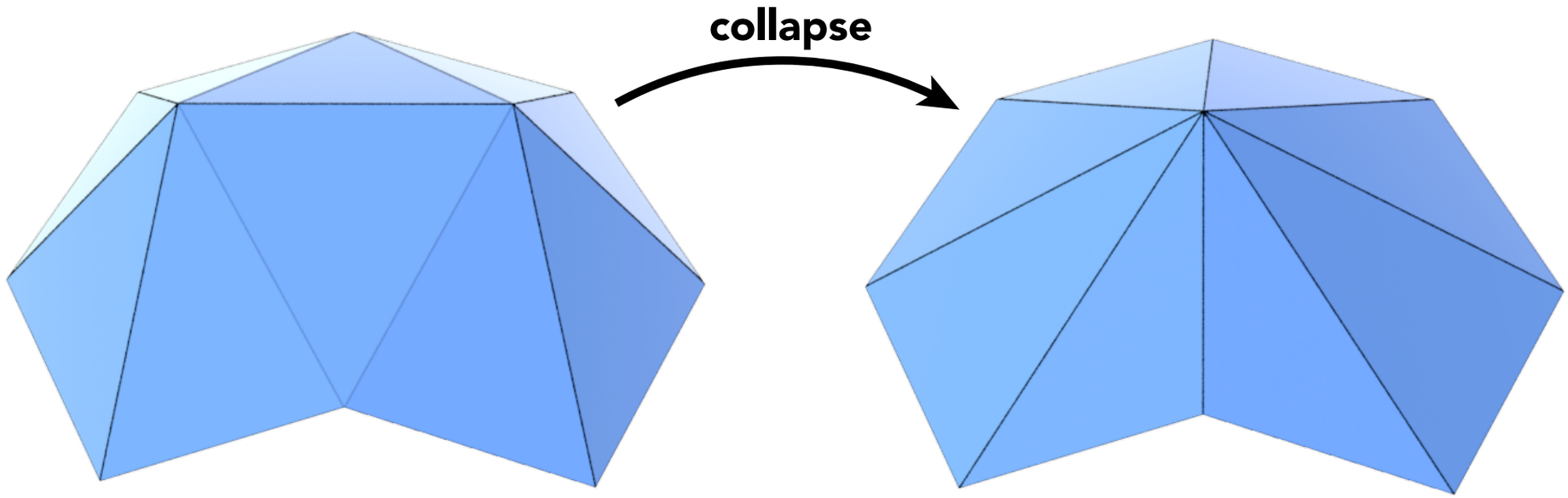
- How much geometric error is introduced by simplification?
- Not a good idea to perform local averaging of vertices
- Quadric error: new vertex should minimize its sum of square distance (L2 distance) to previously related triangle planes!



http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf

Quadric Error of Edge Collapse

- How much does it cost to collapse an edge?
- Idea: compute edge midpoint, measure quadric error



- Better idea: choose point that minimizes quadric error
- More details: Garland & Heckbert 1997.

Simplification via Quadric Error

Iteratively collapse edges

Which edges? Assign score with quadric error metric*

- approximate distance to surface as sum of distances to planes containing triangles
- iteratively collapse edge with smallest score
- greedy algorithm... great results!

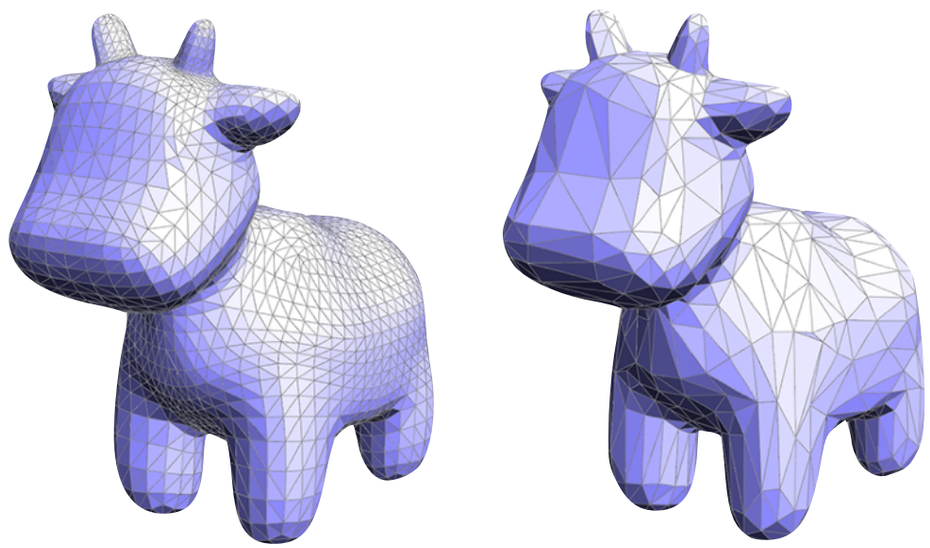
基于二次误差度量，就可以寻找坍缩代价最小的边，用代价最小的坍缩方式进行坍缩

* (Garland & Heckbert 1997)

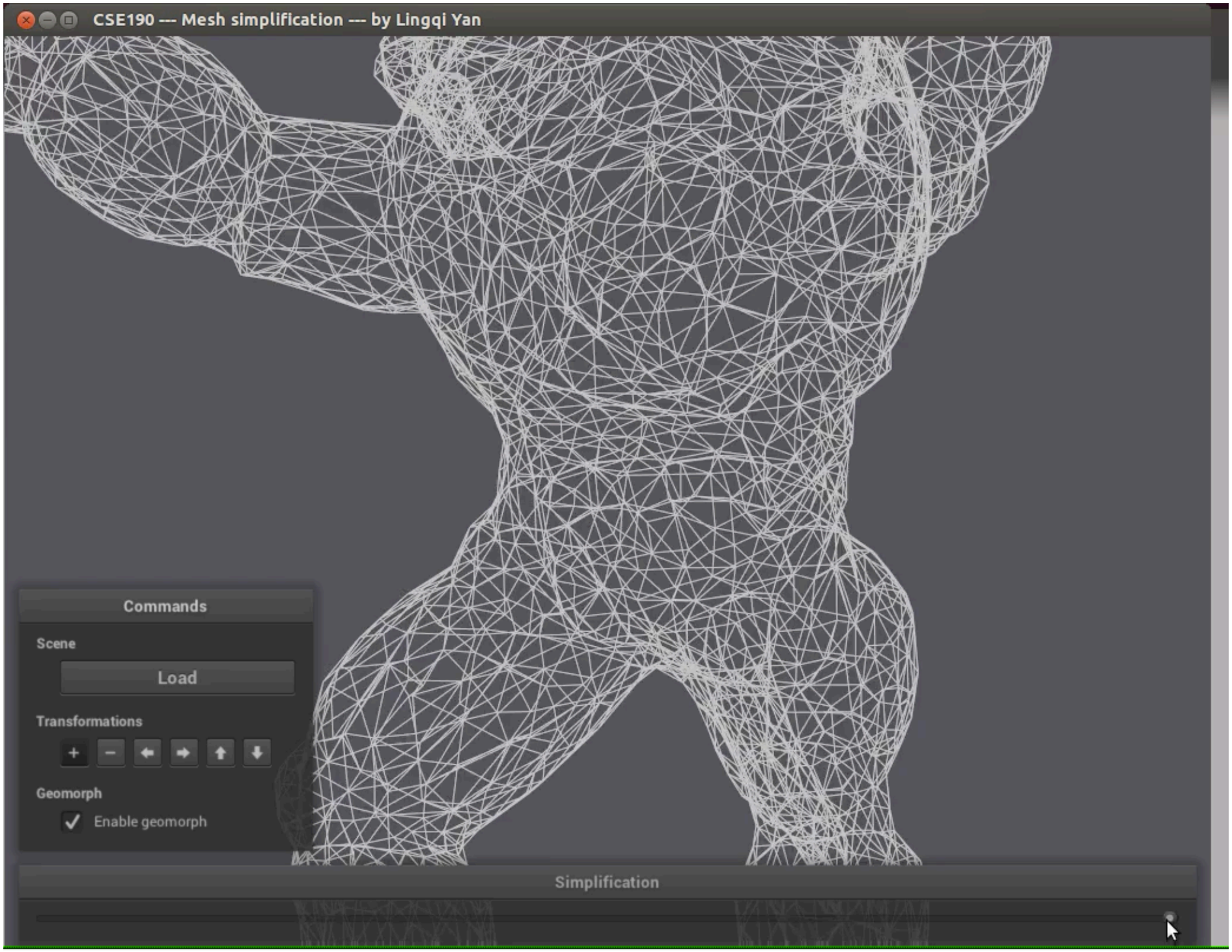
Quadric Error Mesh Simplification

5,804 **994** **532** **248** **64**

Garland and Heckbert '97



Quadric Error Mesh Simplification



Before we move on…

- Shadows
  - How to draw shadows using rasterization?
  - **Shadow mapping**!

Shadow of the Tomb Raider, 2018

Shadow Mapping

光栅化由于其局部性问题，想直接实现阴影效果是不行的，需要一些额外的方法。
Shadow Mapping 就是这样的一种普遍的处理方式。不过 Shadow Mapping 只能处理点光源的问题，但对一般的游戏而言通常是足够的。
核心思路是：阴影是那些能直达摄像机，但不能直达光源的像素点。

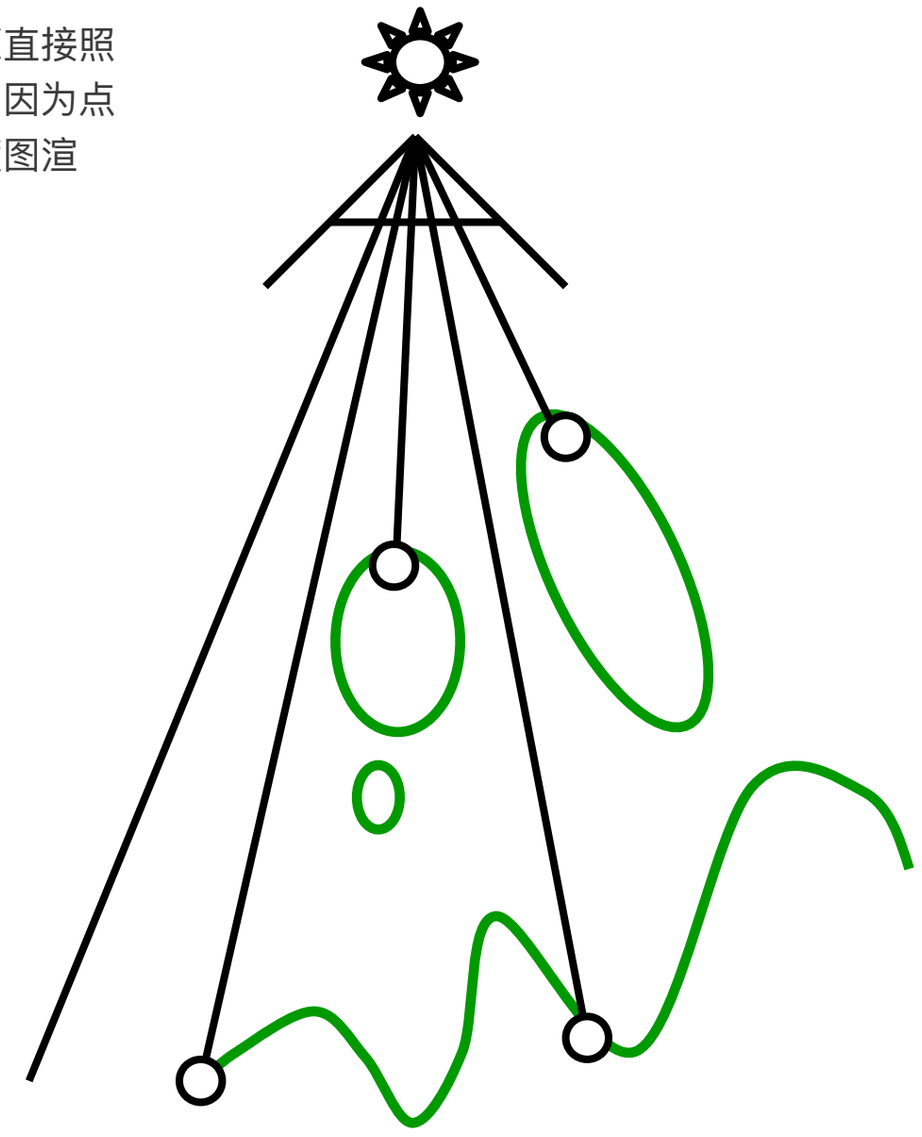
- An Image-space Algorithm
  - no knowledge of scene’s geometry during shadow computation
  - must deal with aliasing artifacts

- Key idea:
  - the points NOT in shadow must be seen both **by the light** and **by the camera**

Pass 1: Render from Light

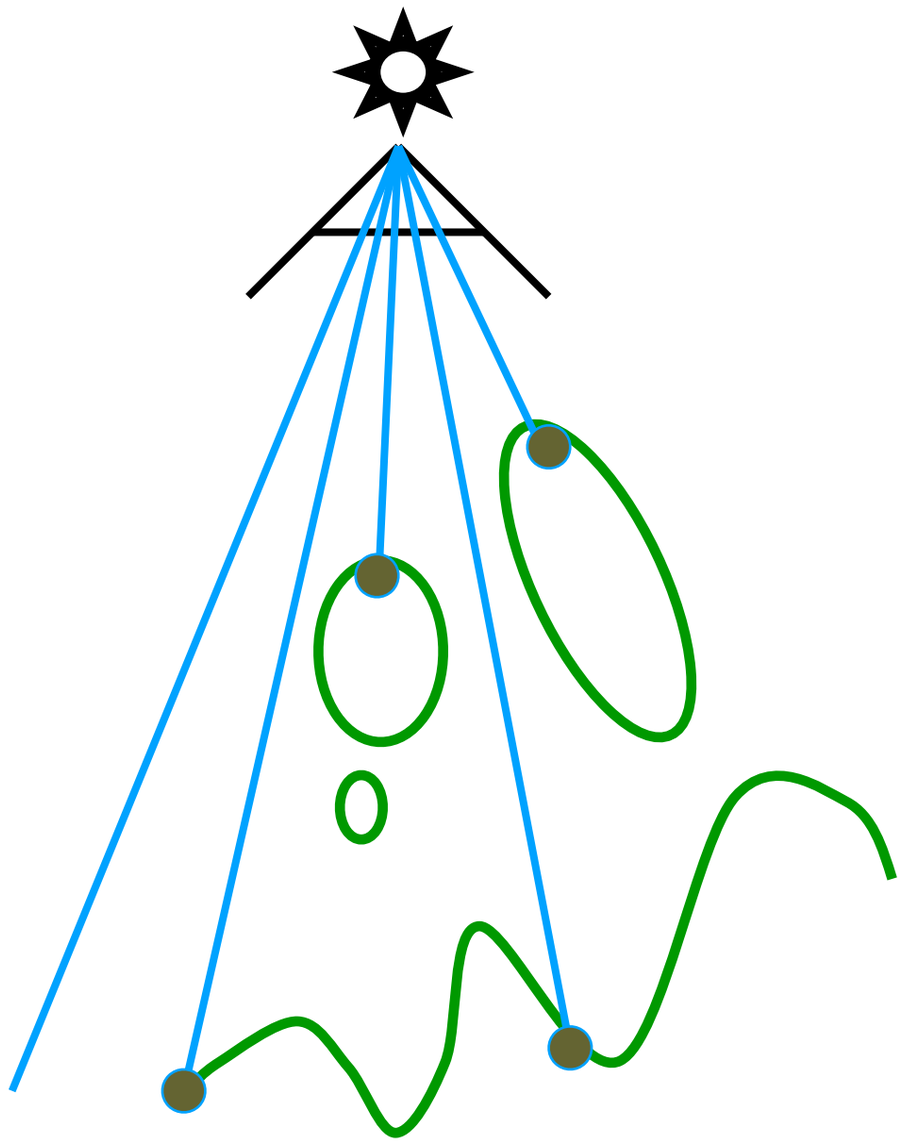
- Depth image from light source

首先将摄像机放在光源位置，看看哪些点能被光源直接照亮。但记录哪些点可以被照亮是一件很难的事情，因为点是可以无限稠密的。所以一种方式是进行一次深度图渲染。用于保存光源看到的深度信息。



Pass 1: Render from Light

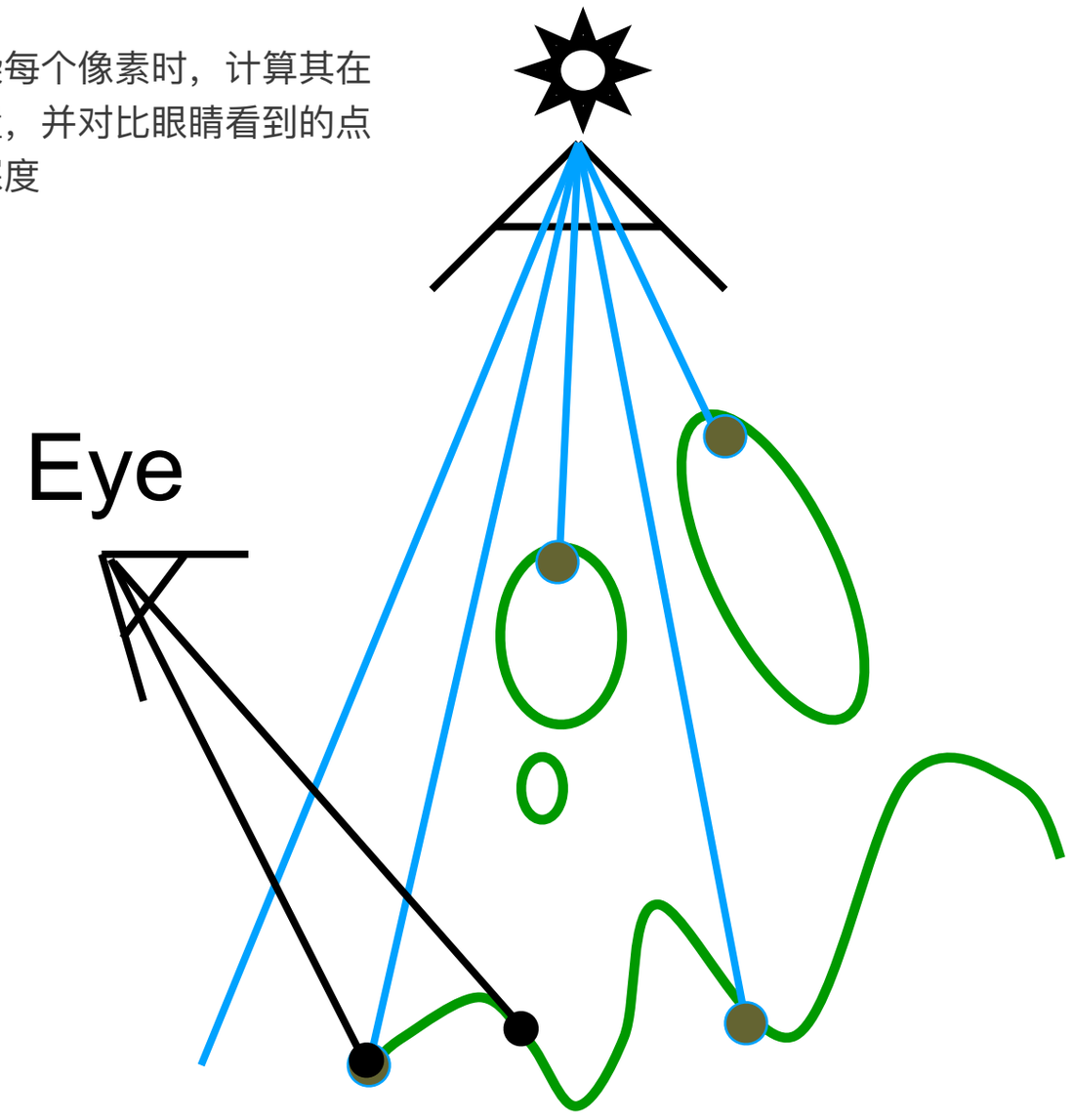
- Depth image from light source



Pass 2A: Render from Eye

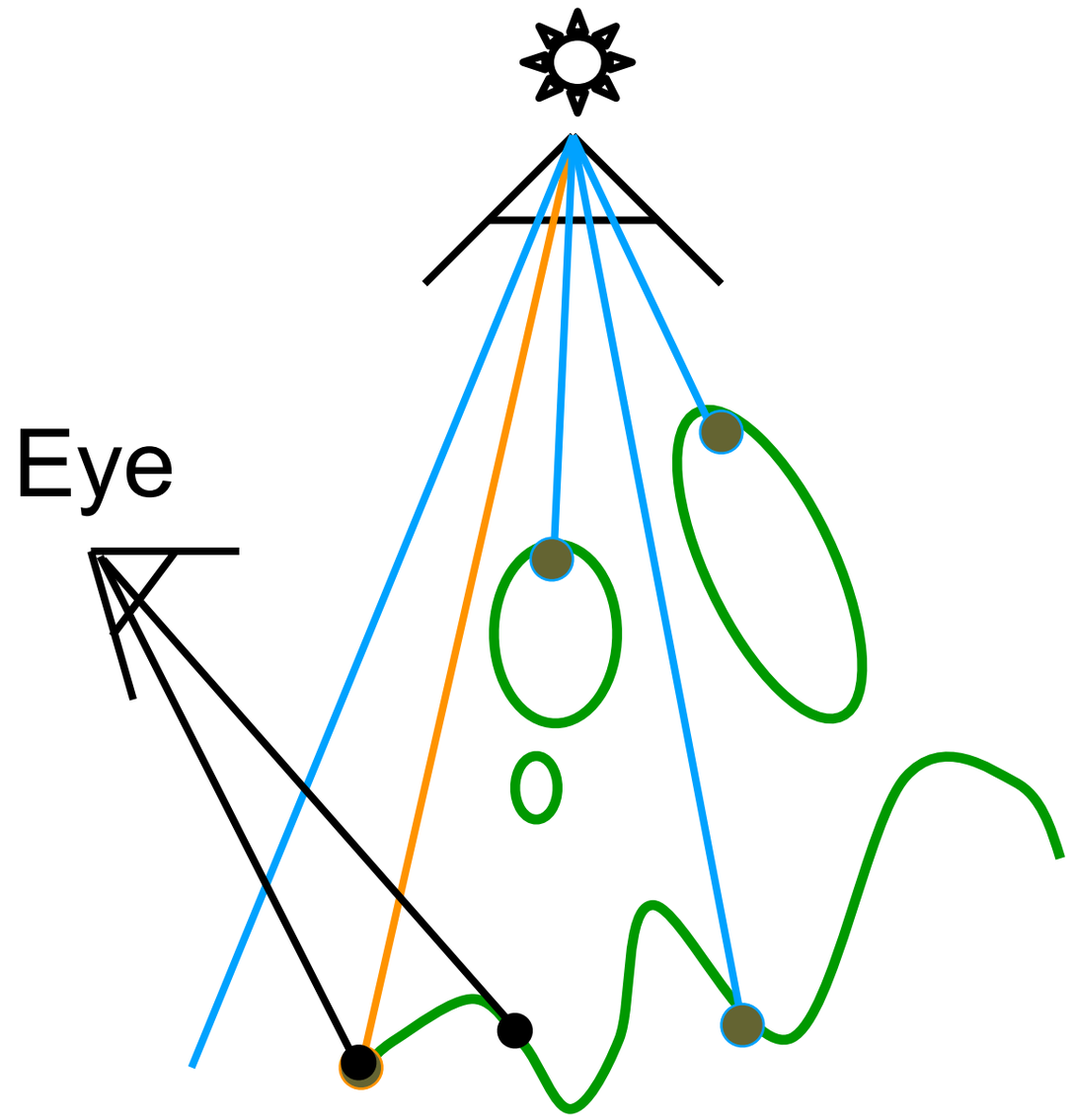
- Standard image (with depth) from eye

再从眼睛的位置看向场景，渲染每个像素时，计算其在之前光源的深度图中的像素位置，并对比眼睛看到的点的光源深度和之前记录的光源深度



Pass 2B: Project to light

- Project visible points in eye view back to light source

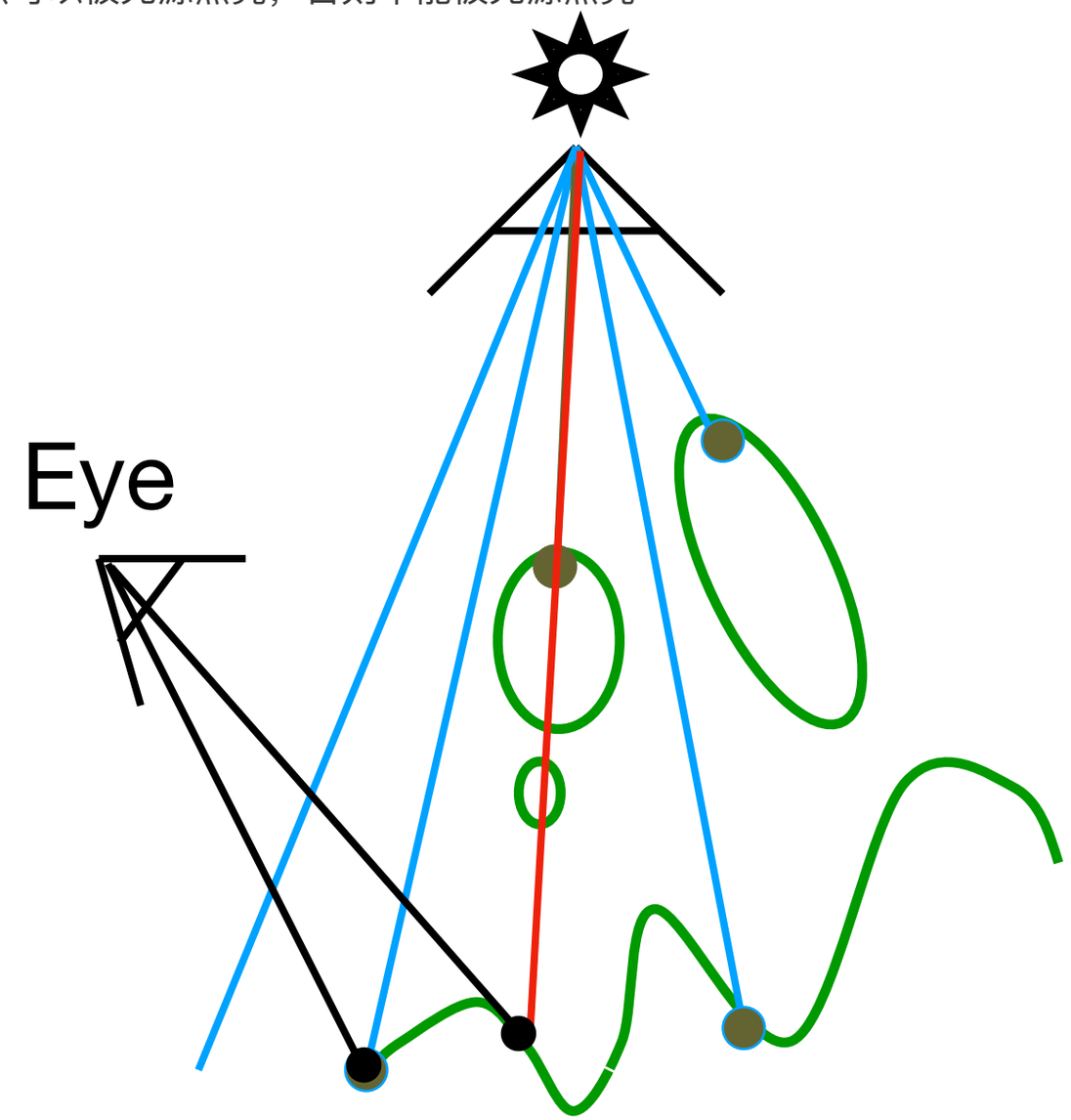


(Reprojected) depths match for light and eye. VISIBLE

Pass 2B: Project to light

- Project visible points in eye view back to light source

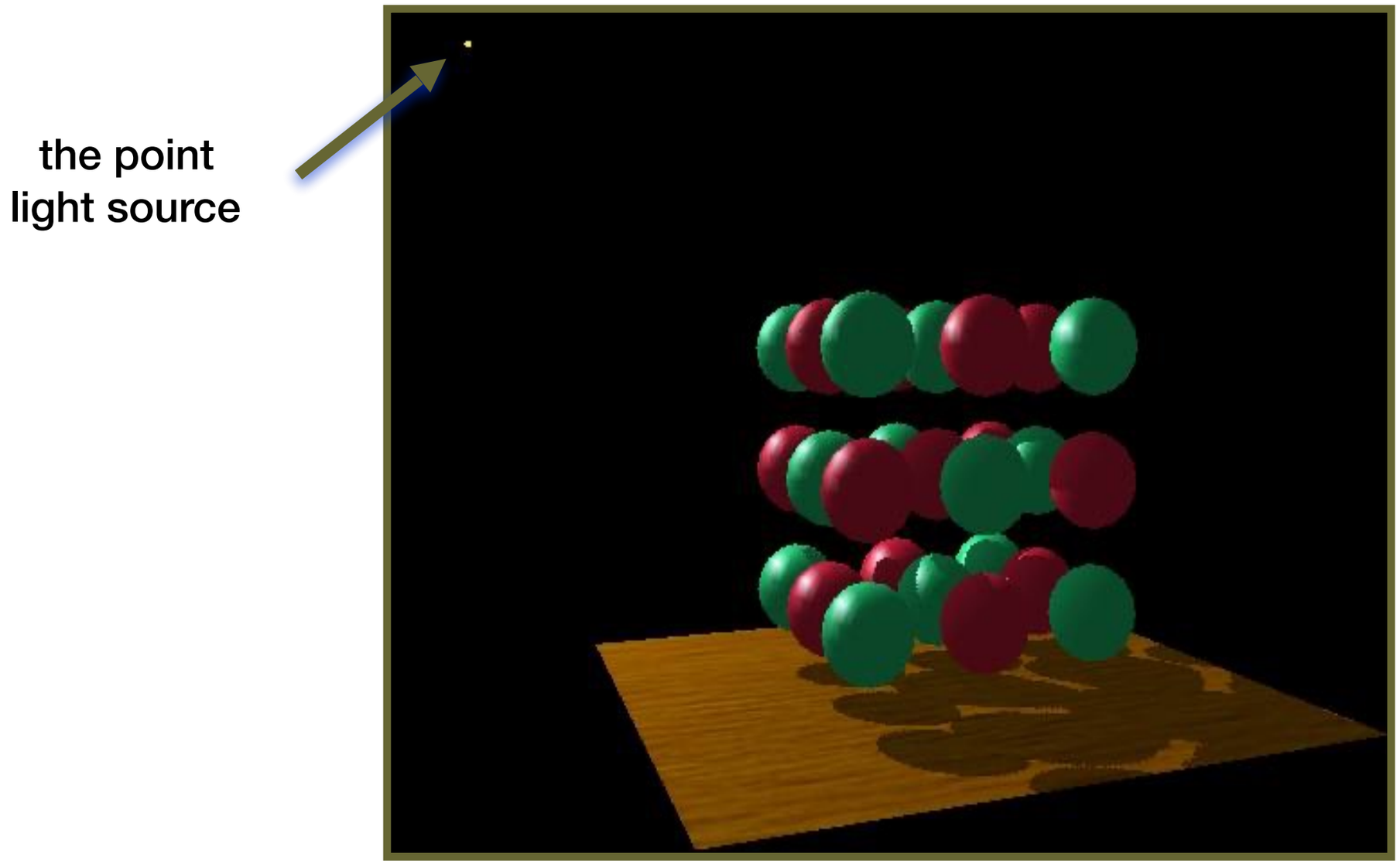
如果两者深度一致则意味着这个点可以被光源照亮，否则不能被光源照亮



(Reprojected) depths from light and eye are not the same. BLOCKED!!

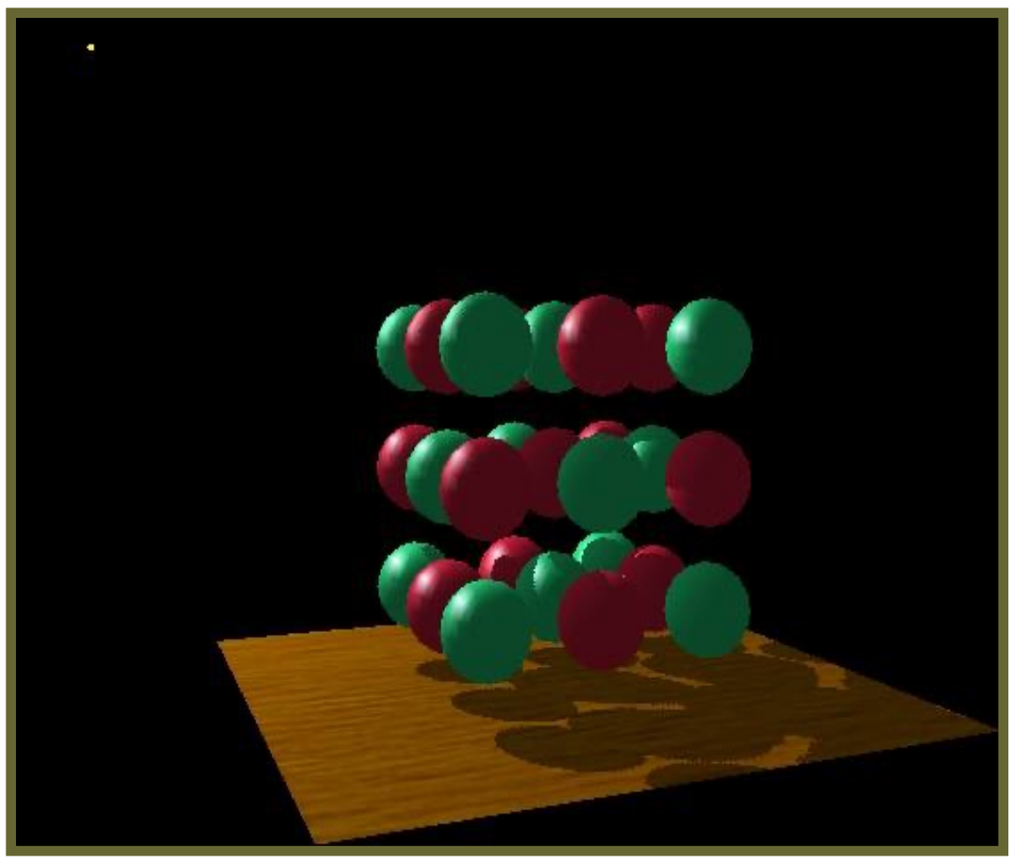
Visualizing Shadow Mapping

- A fairly complex scene with shadows

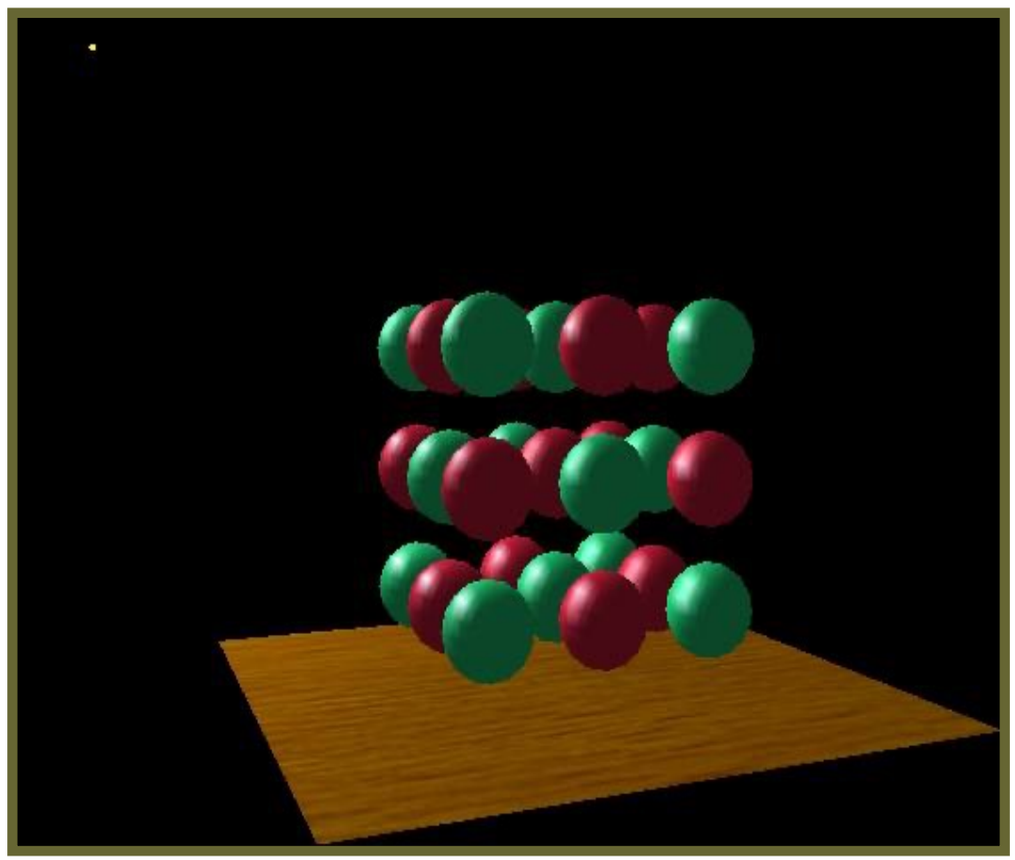


Visualizing Shadow Mapping

- Compare with and without shadows



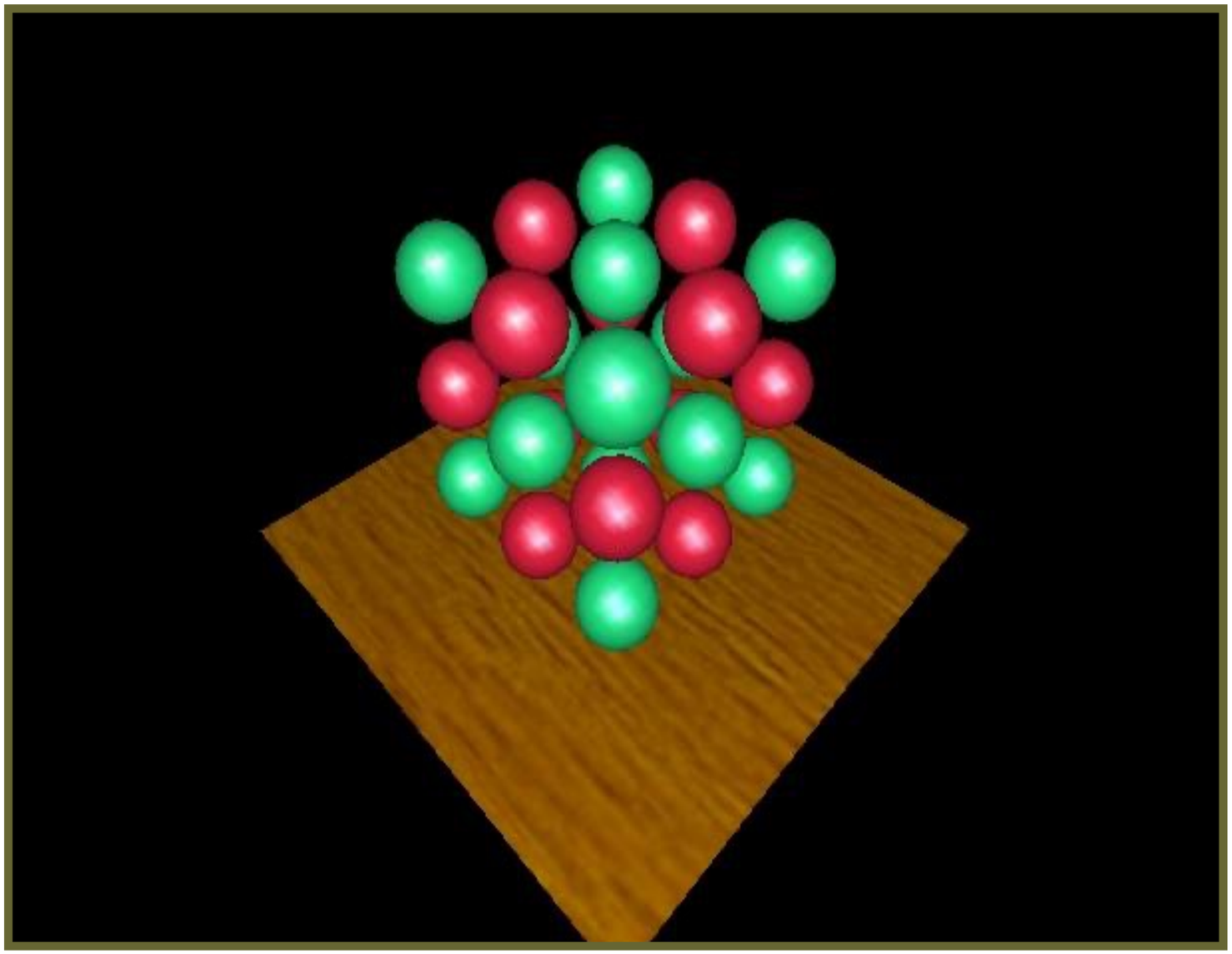
with shadows

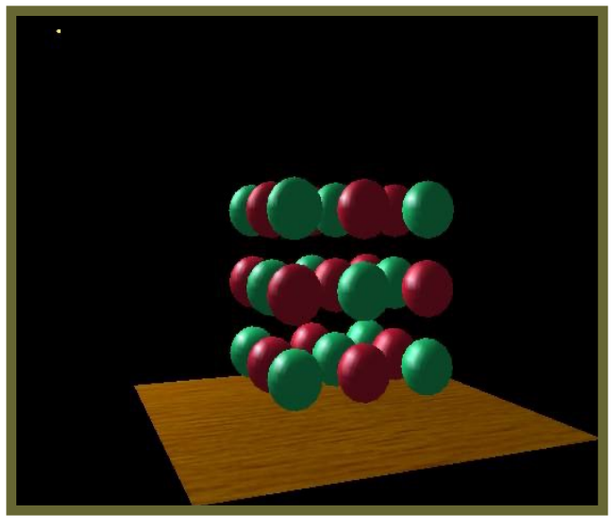


without shadows

Visualizing Shadow Mapping

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

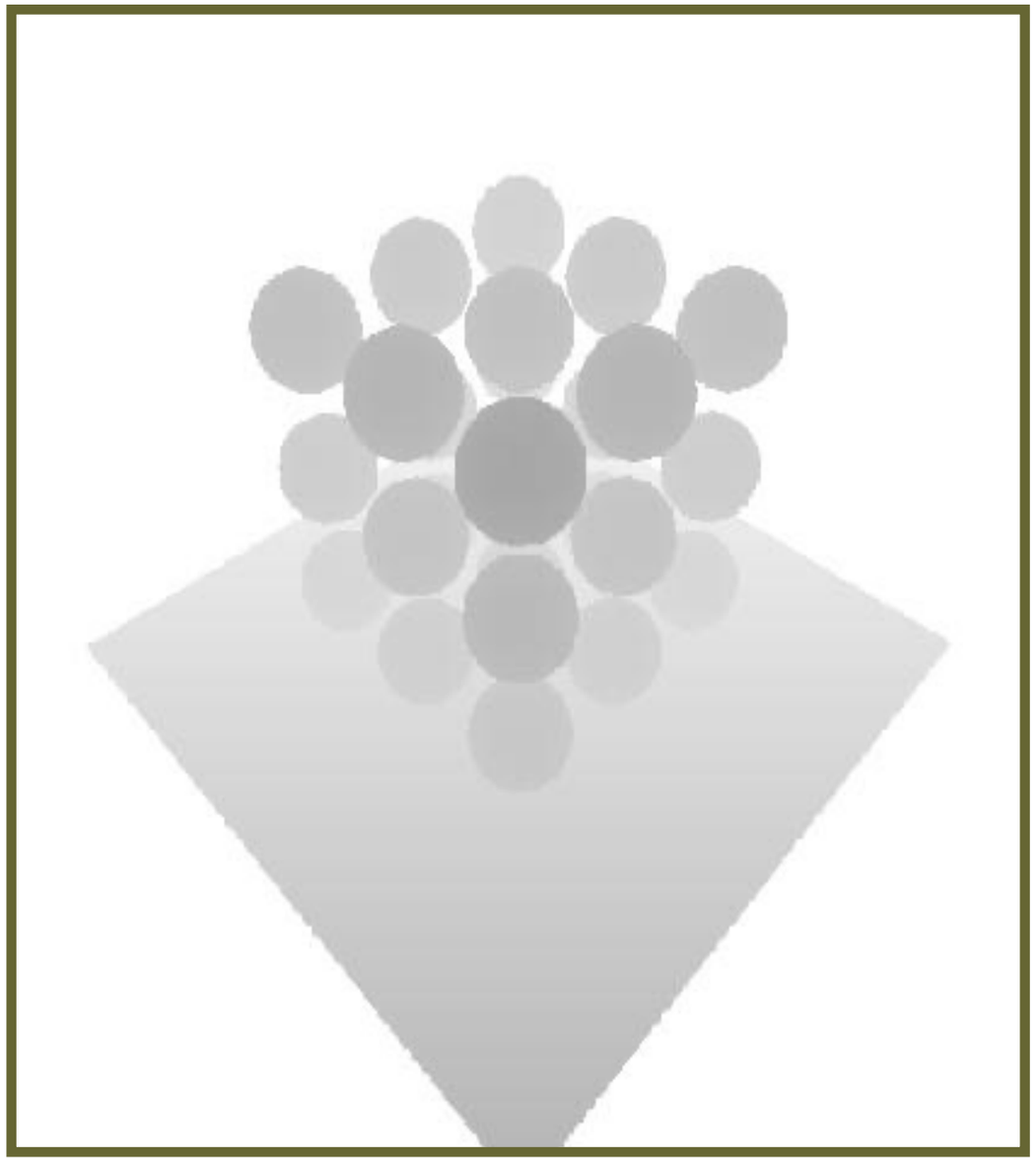


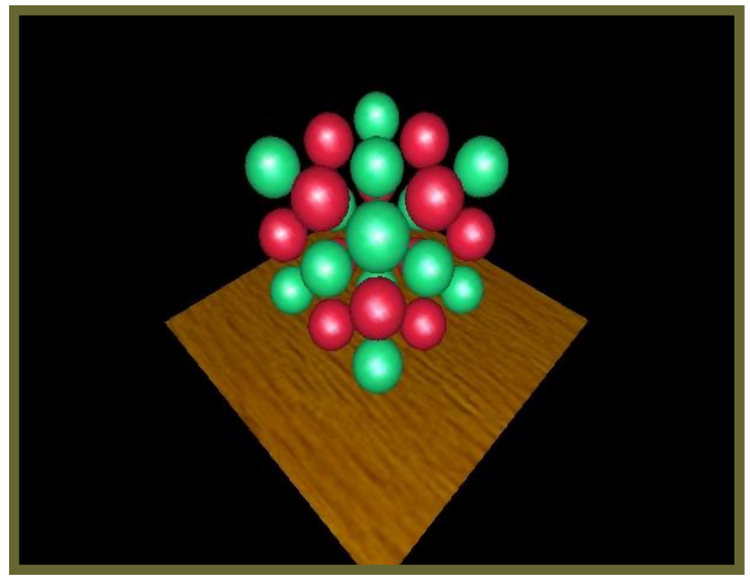


FYI: from the eye’s point-of-view again

Visualizing Shadow Mapping

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



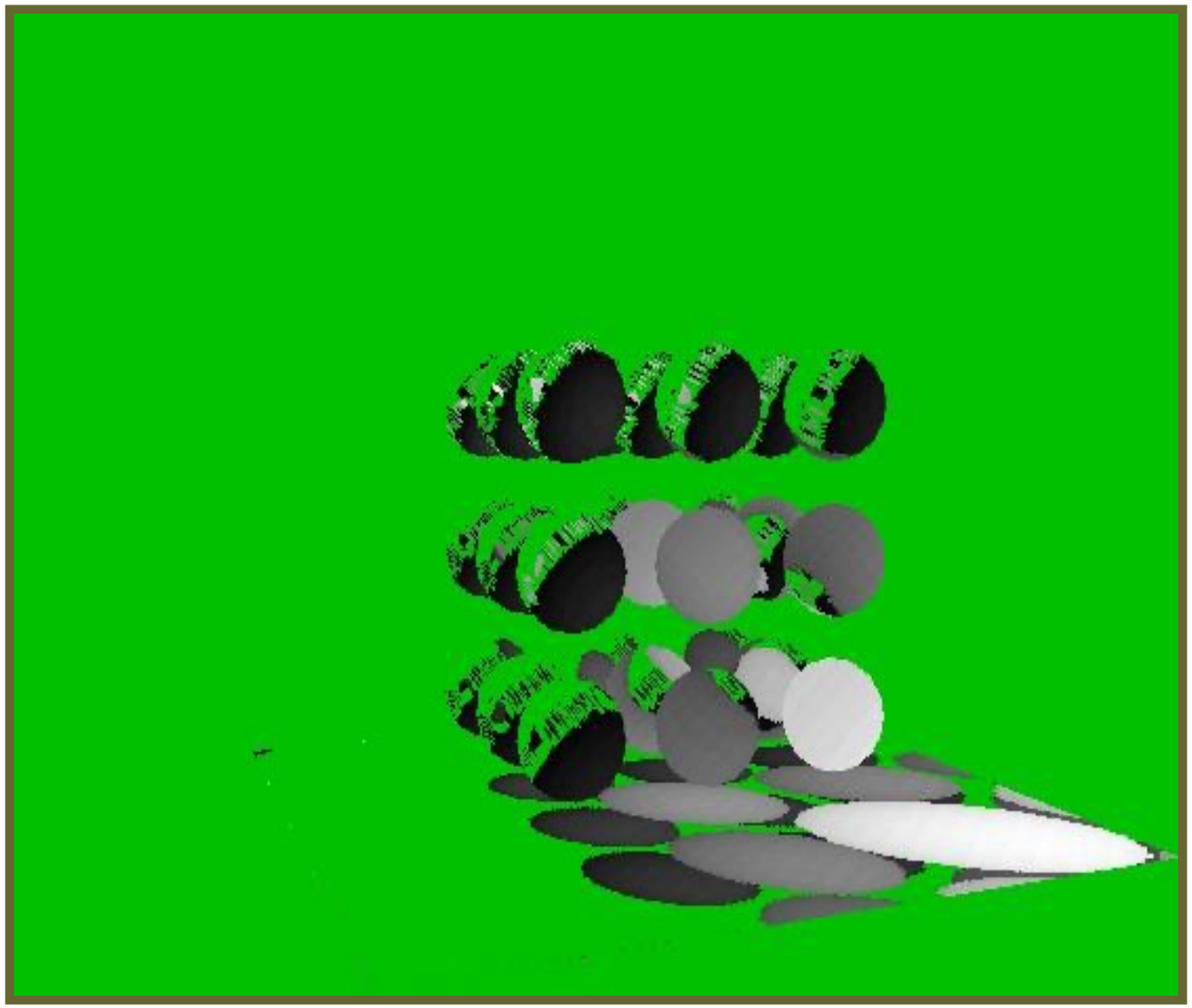


FYI: from the light’s point-of-view again

Visualizing Shadow Mapping

- Comparing Dist(light, shading point) with shadow map

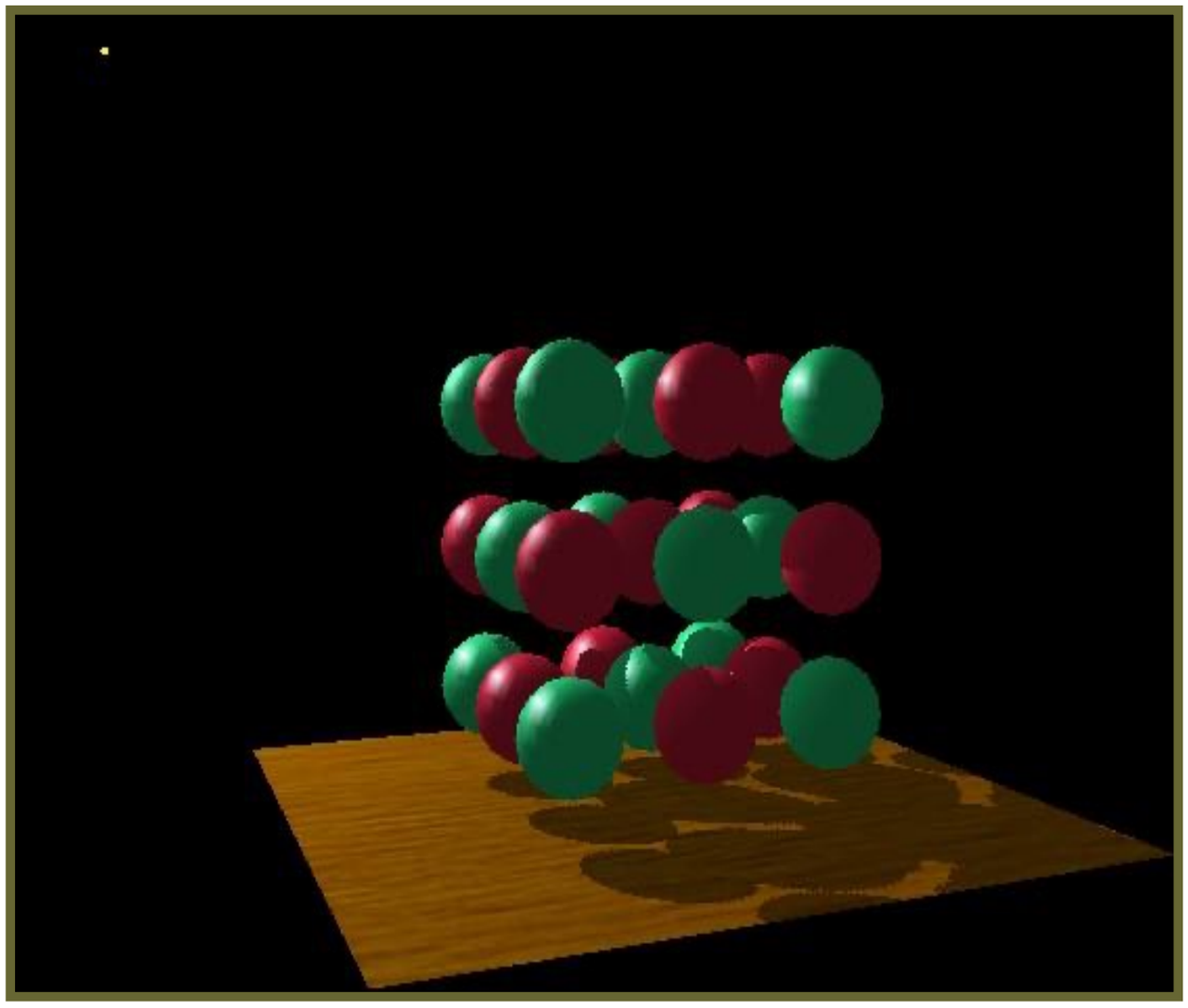
Green is where the distance(light, shading point) ≈ depth on the shadow map



Non-green is where shadows should be

Visualizing Shadow Mapping

- Scene with shadows



Shadow Mapping

- Well known rendering technique
  - Basic shadowing technique for early animations (Toy Story, etc.) and in EVERY 3D video game

Zelda: Breath of the Wild

Super Mario Odyssey

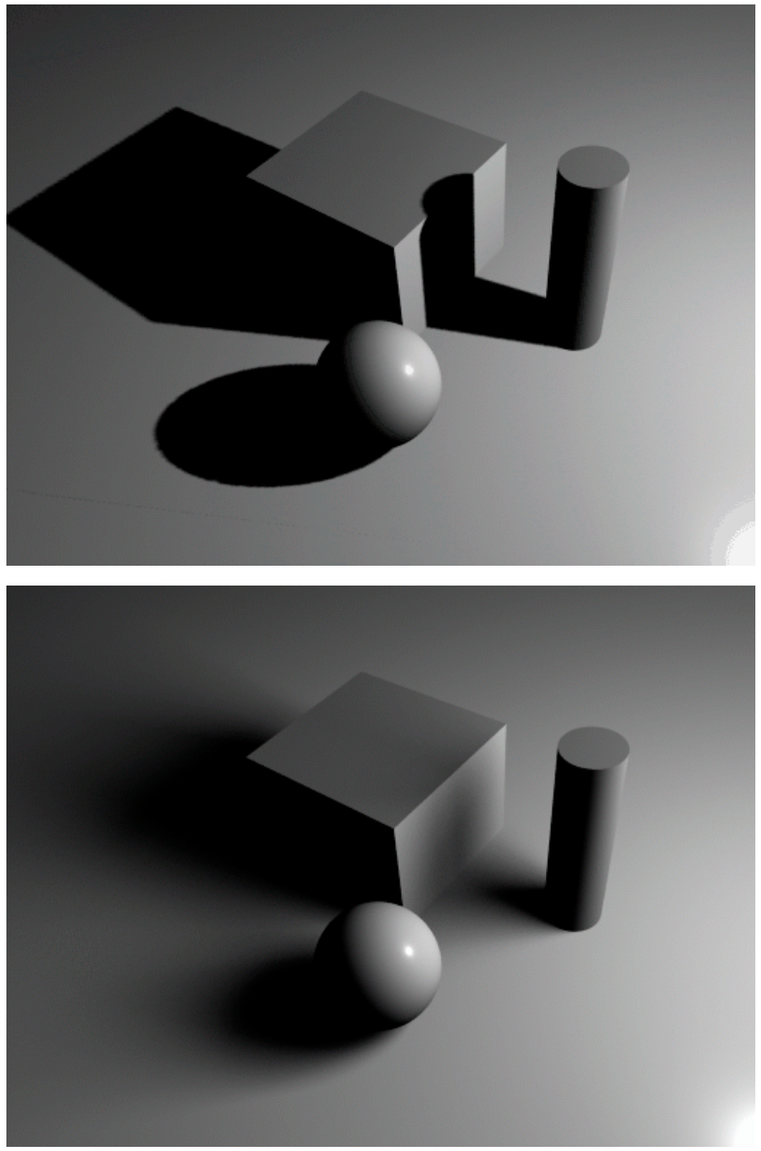
Problems with shadow maps

- Hard shadows (point lights only)
- Quality depends on shadow map resolution (general problem with image-based techniques)
- Involves equality comparison of floating point depth values means issues of scale, bias, tolerance

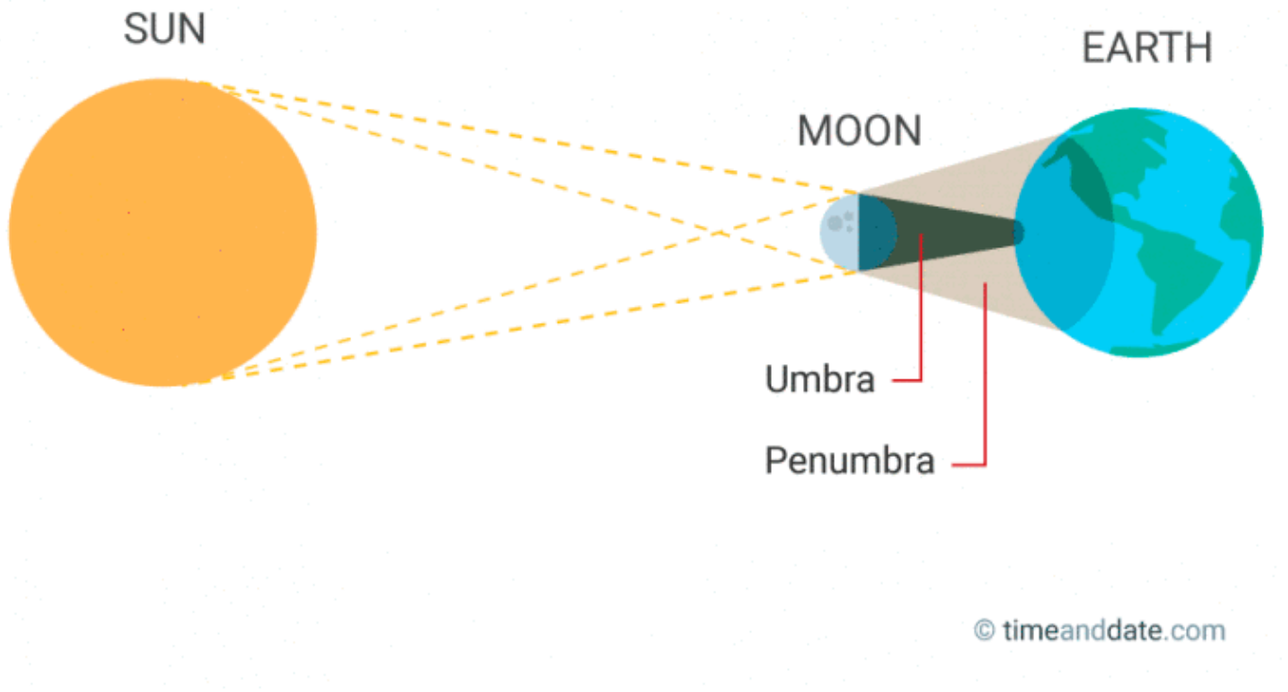
由于 Shadow Mapping 只能处理点光源。于是只能得到硬阴影。而现实世界的光源往往是面光源，因而通常是软阴影，这使 Shadow Mapping 一定程度上不够真实。
要处理这个问题则需要光追算法。

Problems with shadow maps

- Hard shadows vs. soft shadows



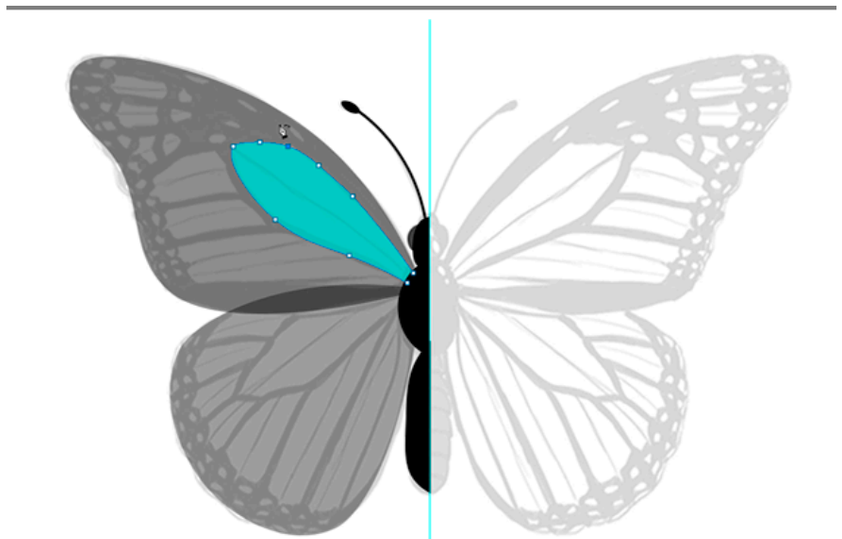
[RenderMan]



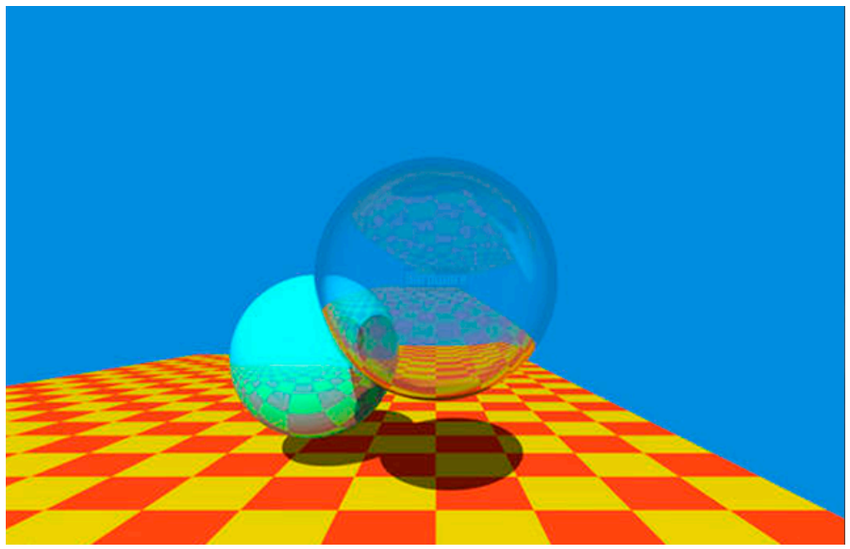
[https://www.timeanddate.com/eclipse/umbra-shadow.html]

Course Roadmap

Rasterization



Geometry



Ray tracing

Animation / simulation

Thank you!

(And thank Prof. Ravi Ramamoorthi and Prof. Ren Ng for many of the slides!)