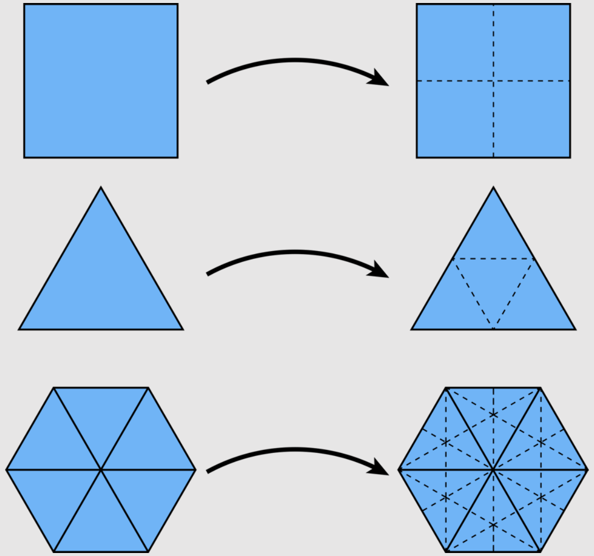
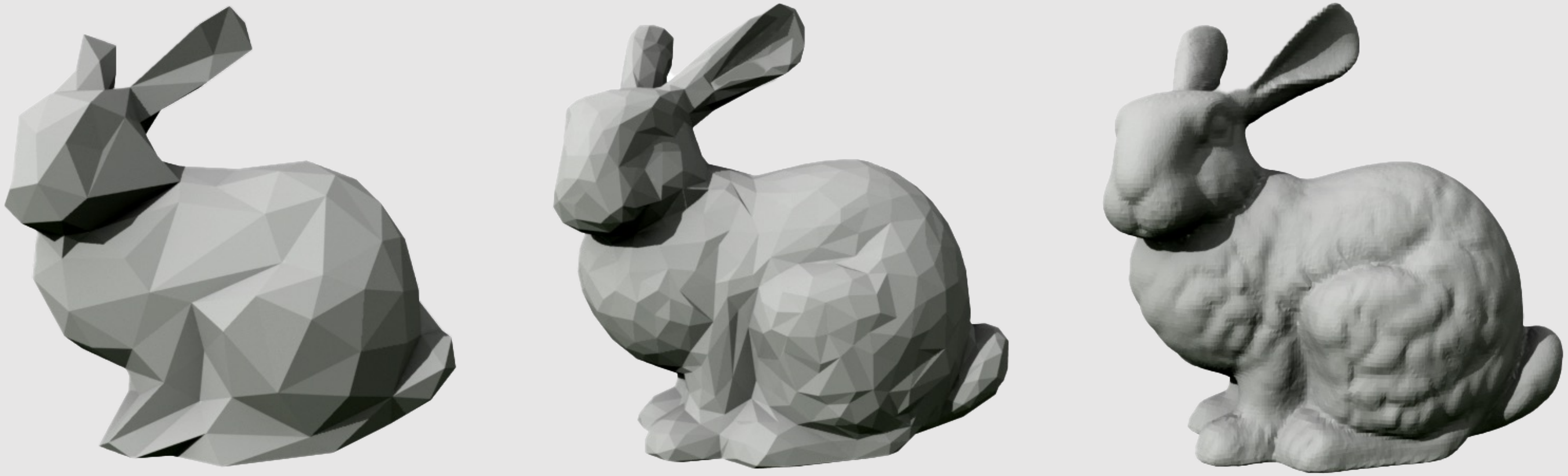
More Digital Geometric Processing

- Digital Geometric Processing
    - Geometric Subdivision
    - Geometric Simplification
    - Geometric Remeshing
    - Geometric Queries

Subdivision

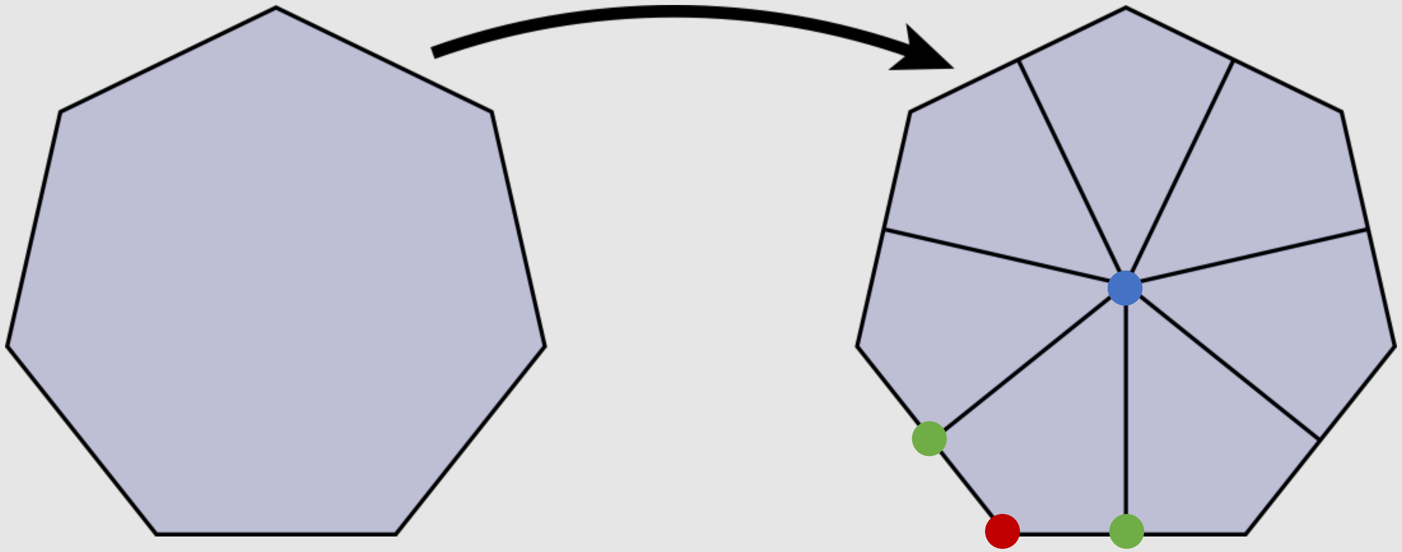
- Subdivison is the process of **upsampling** a mesh
- General formula:
  - **Split Step:** split faces into smaller faces
  - **Move Step:** replace vertex positions/properties with weighted average of neighbors





Linear Subdivision [Split Step]

- Split every polygon (any # of sides) into quadrilaterals

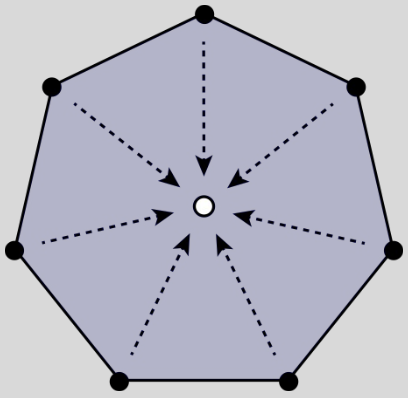


- Each new quadrilateral now has:
  - [face coords] : 1 new vertex from the mesh face center
  - [edge coords] : 2 new vertices from the new edges
  - [vertex coords] : 1 new vertex from the original mesh face

Linear Subdivision [Move Step]

Step 1:

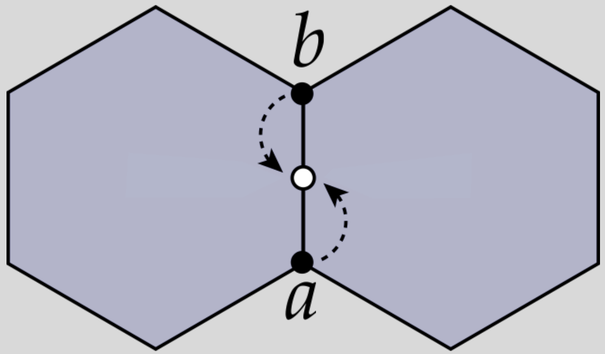
Face Coords



$$\frac{1}{n}\sum_i p_i$$

Step 2:

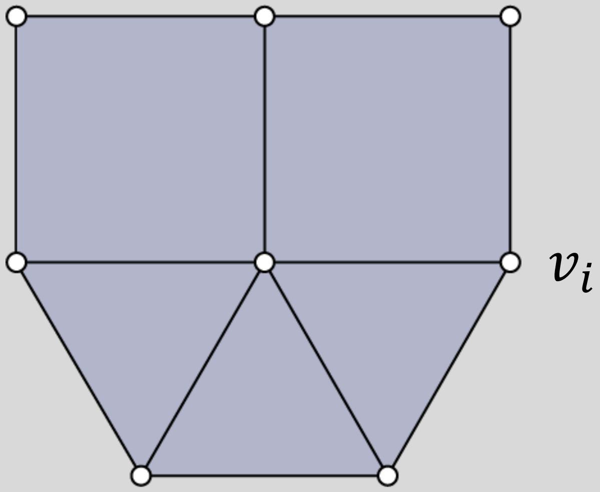
Edge Coords



$$(a + b) \;/\; 2$$

Step 3:

Vertex Coords

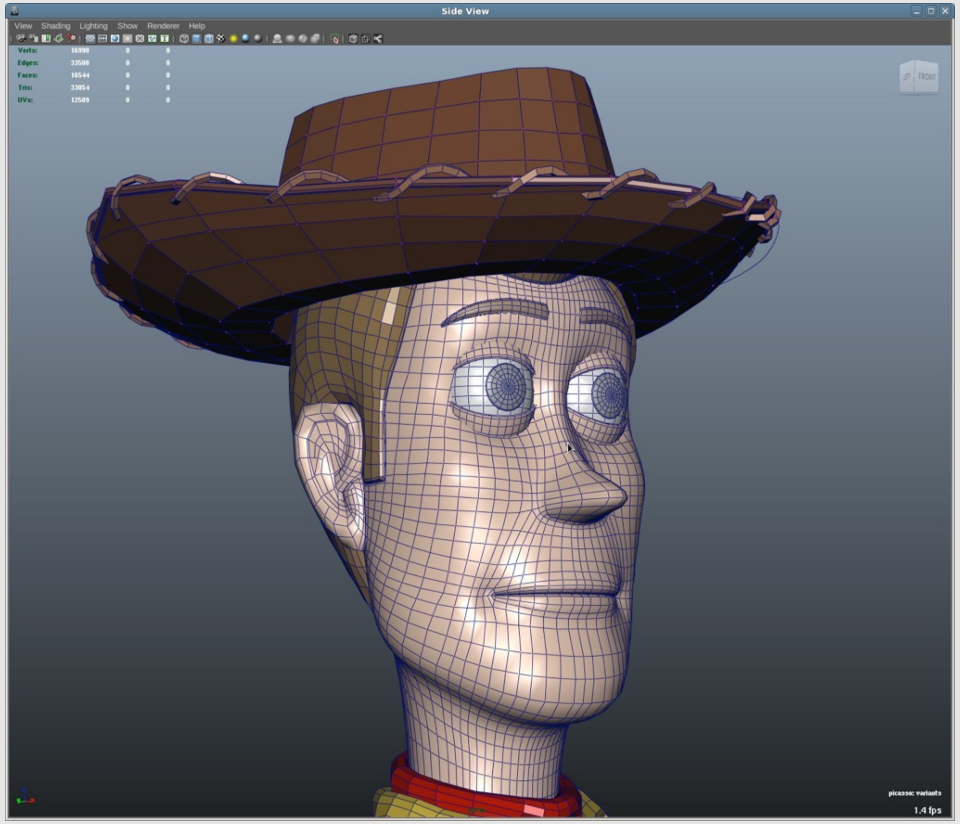


$$v_i = v_i$$

Catmull Clark Subdivision

- In 1978, Edwin Catmull (Pixar co-founder) and Jim Clark wanted to create a generalization of **uniform bi-cubic b-splines** for 3D meshes
  - We will cover what this means in a future lecture :)

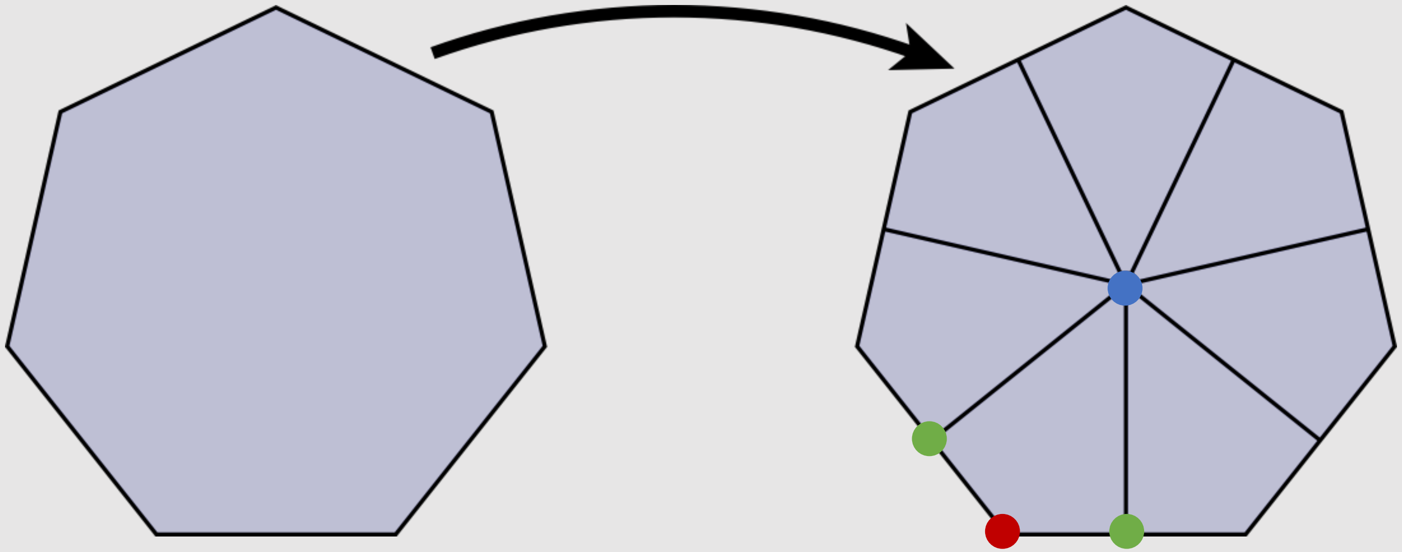
- Became ubiquitous in graphics
  - Helped Catmull win an Academy Award for Technical Achievement in 2005



OpenSubdiv V2 (2018) Pixar

Catmull-Clark Subdivision [Split Step]

- Split every polygon (any # of sides) into quadrilaterals

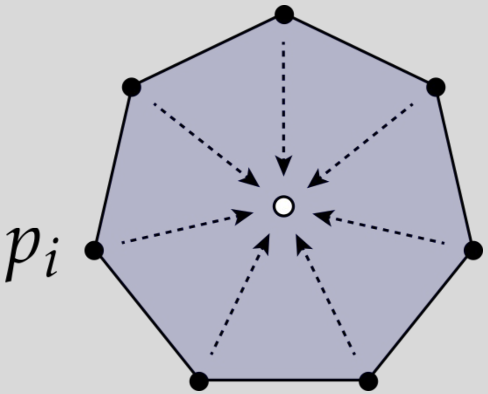


- Each new quadrilateral now has:
  - [face coords] : 1 new vertex from the mesh face center
  - [edge coords] : 2 new vertices from the new edges
  - [vertex coords] : 1 new vertex from the original mesh face

No different than Linear Subdivision!

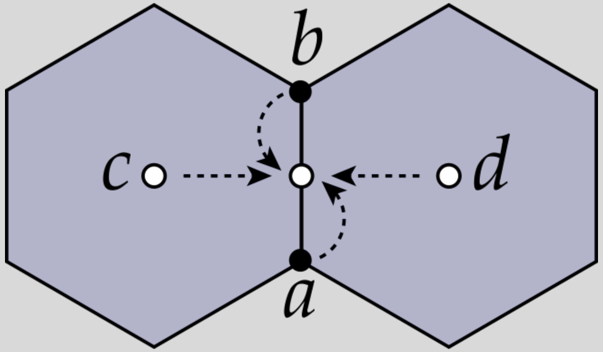
Catmull-Clark Subdivision [Move Step]

Step 1: Face Coords



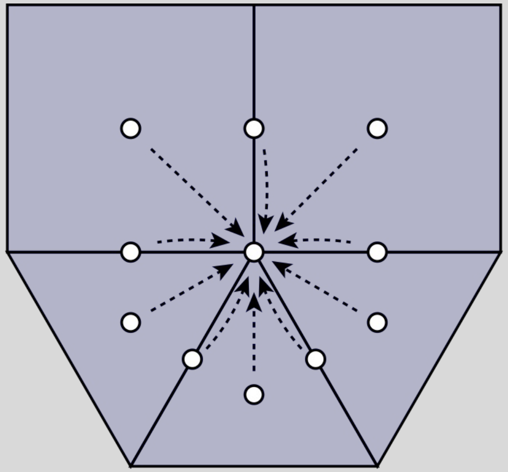
$$\frac{1}{n}\sum_i p_i$$

Step 2: Edge Coords



$$(a+b+c+d)/4$$

Step 3: Vertex Coords



$$\frac{Q + 2R + (n-3)S}{n}$$

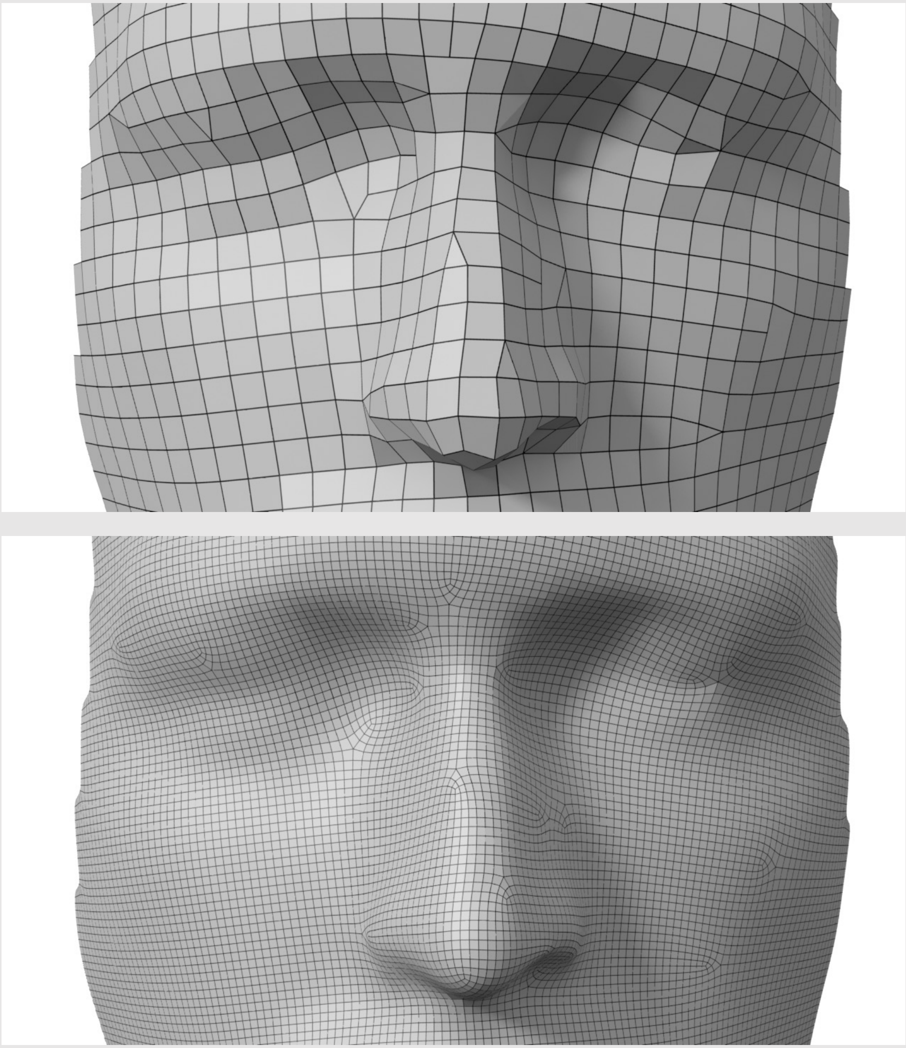
n - vertex degree

Q - average of face coords around vertex

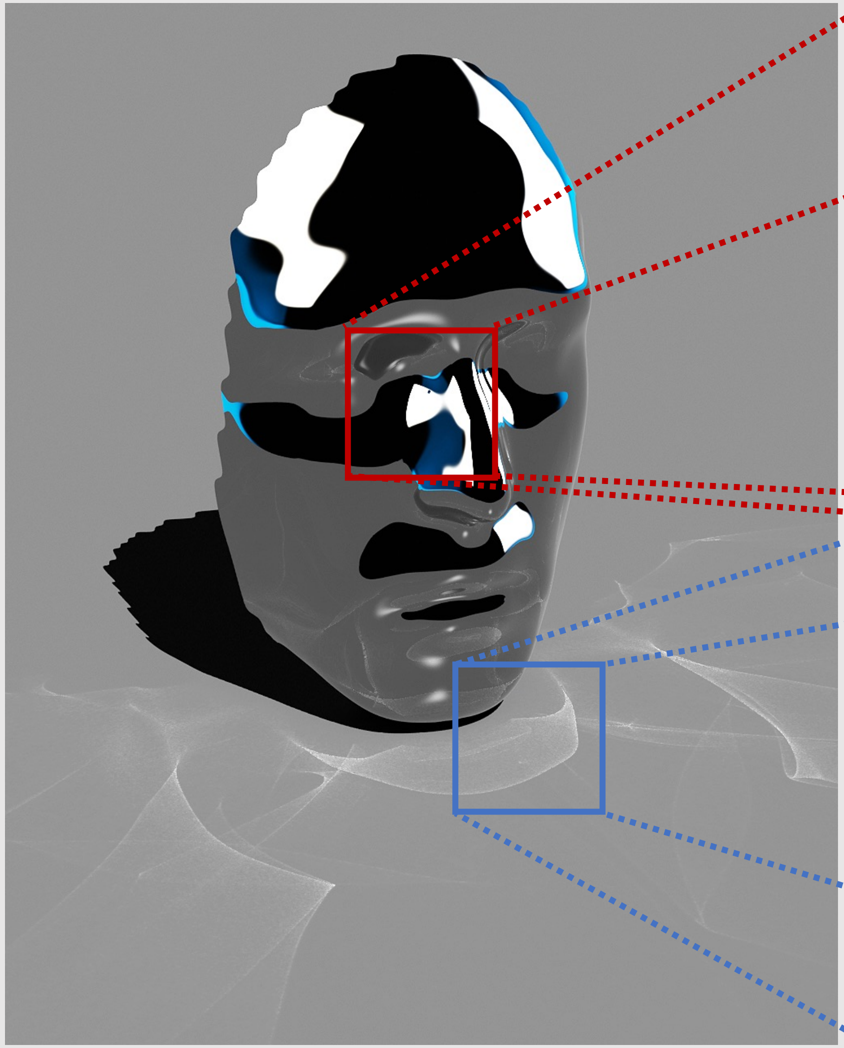
R - average of edge coords around vertex

S - original vertex position

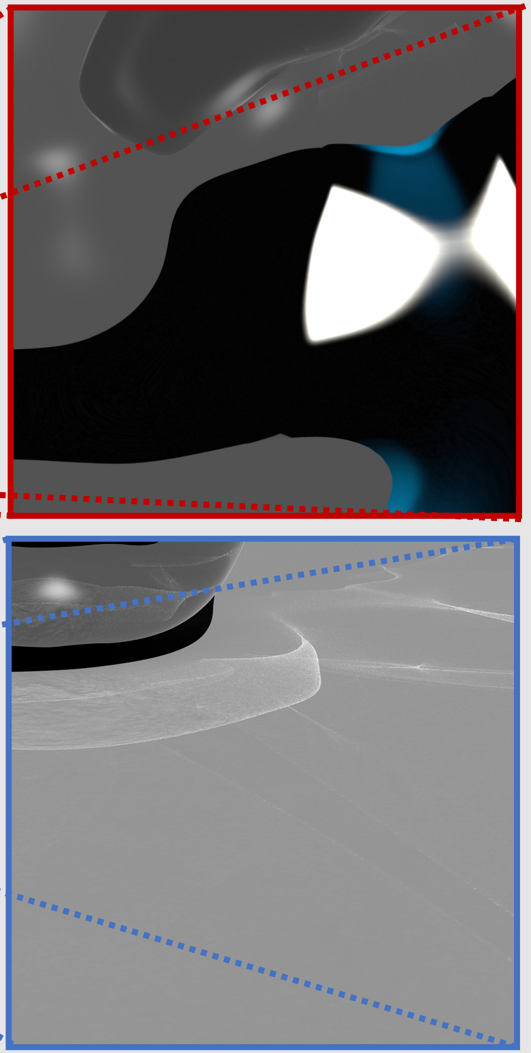
Catmull-Clark Subdivision [Quads]



Few irregular vertices

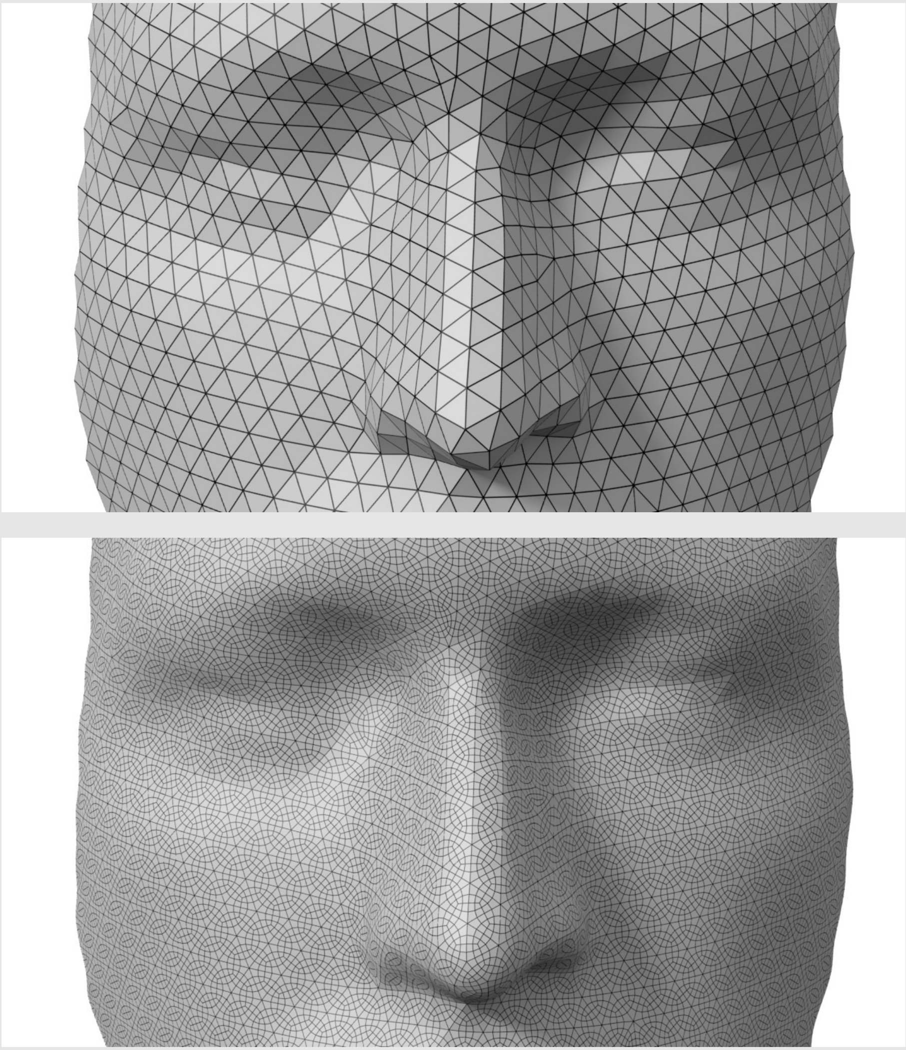


Smoothly-varying surface normals

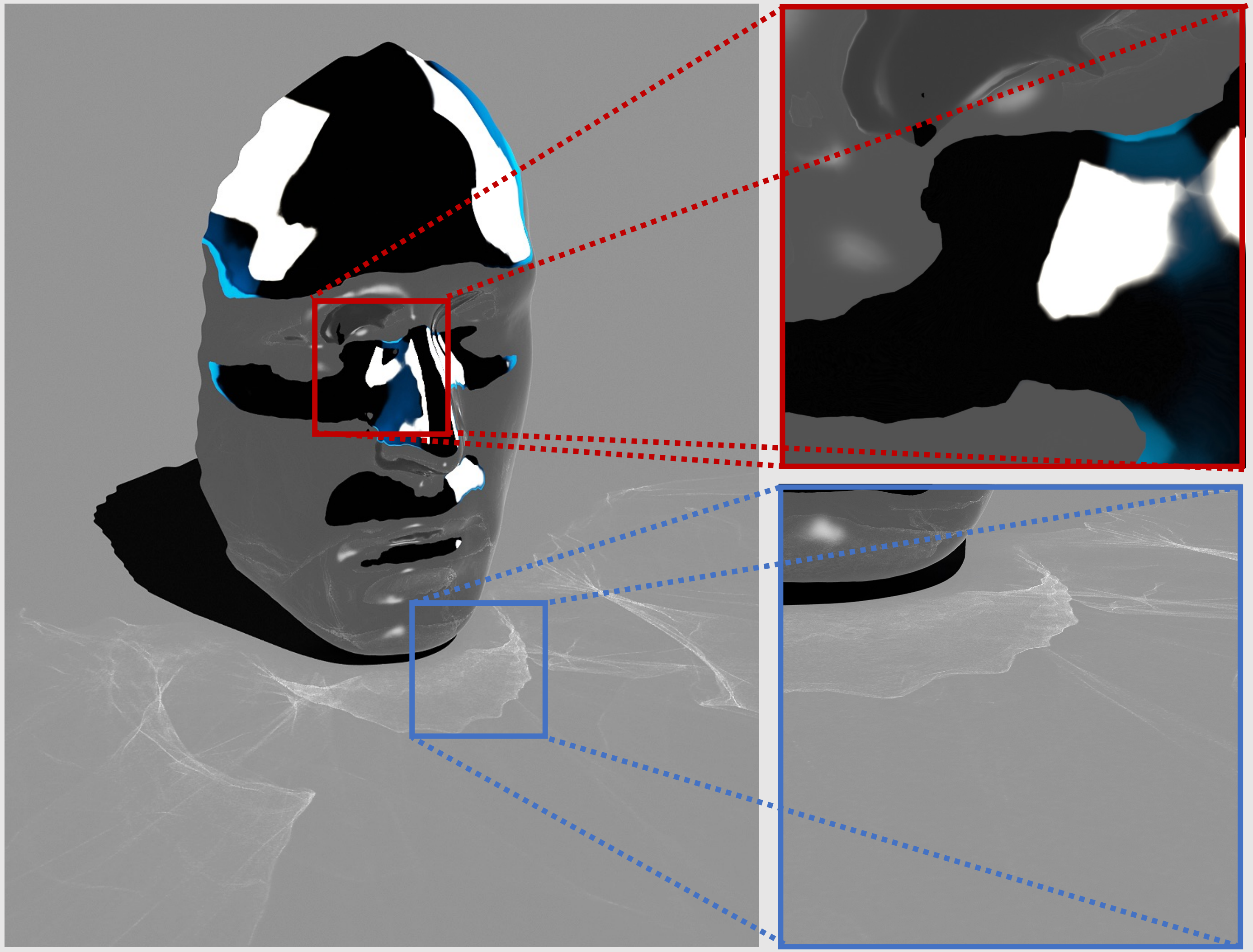


Smooth reflections/caustics

Catmull-Clark Subdivision [Triangles]



Many irregular vertices



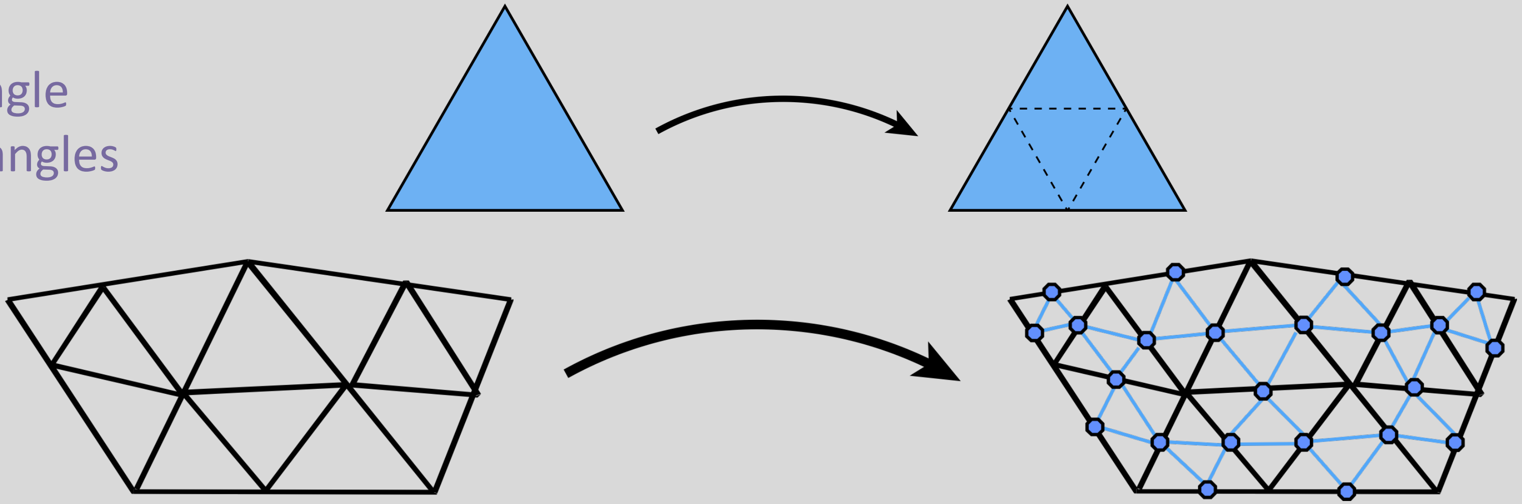
Erratic surface normals

Jagged reflections/caustics

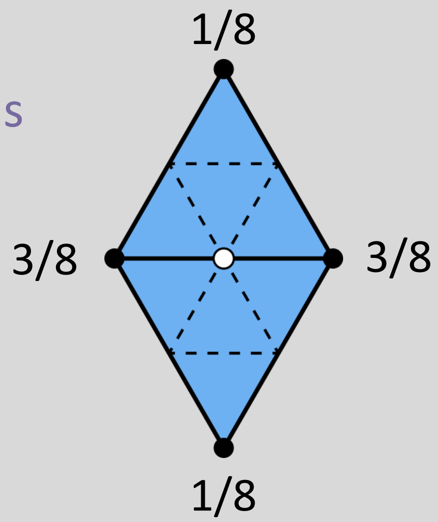
Is there a better subdivision scheme we can use for triangulated meshes?

Loop Subdivision

Step 1:
Split triangle
into 4 triangles

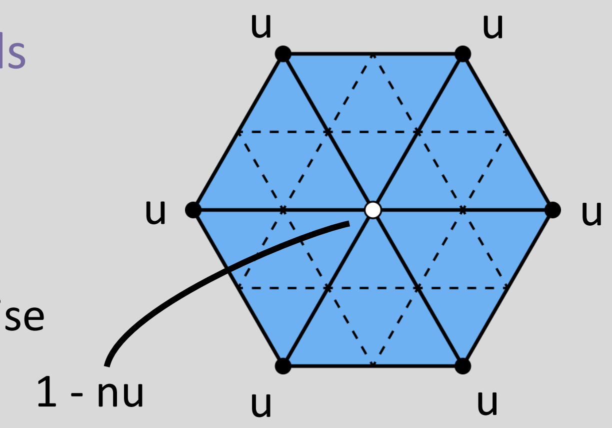


Step 2:
Assign new coords



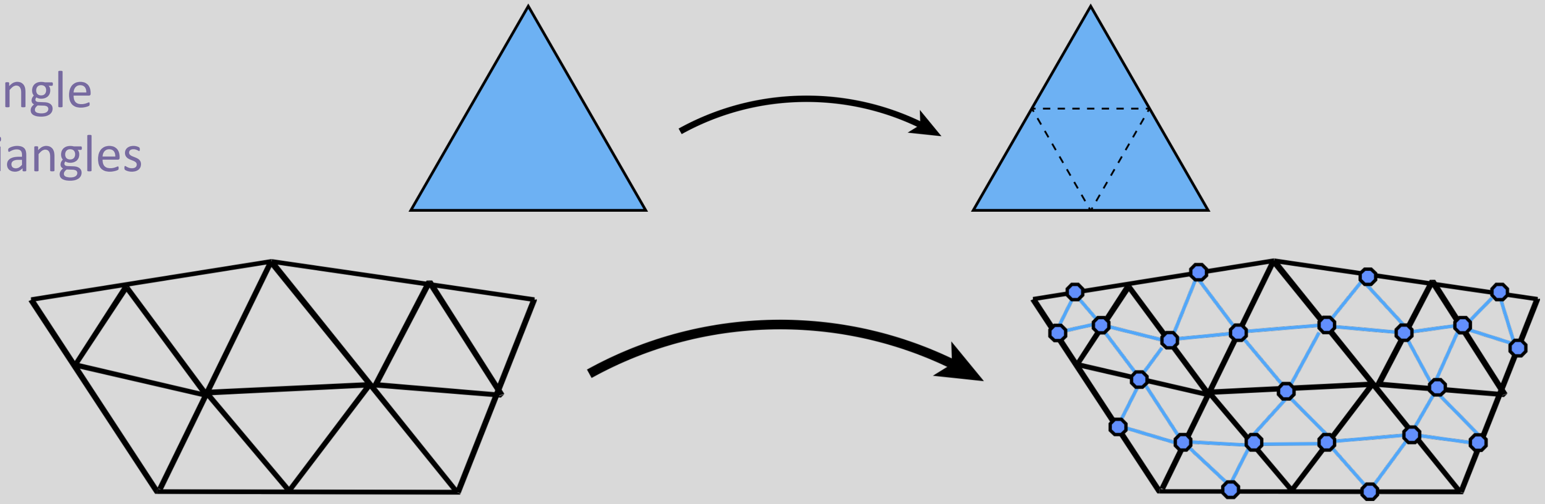
Step 3:
Assign old coords

n - vertex degree
u - 3/16 if n=3
3/(8n) otherwise



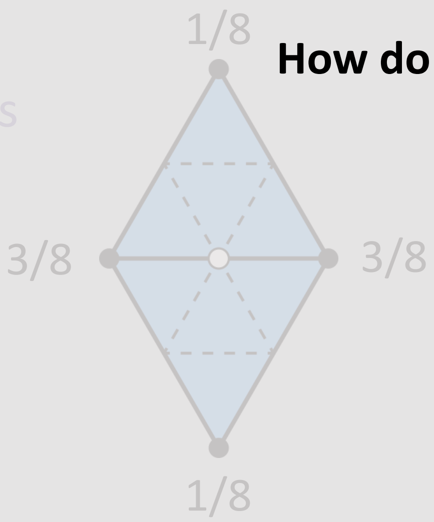
Loop Subdivision

Step 1:
Split triangle
into 4 triangles



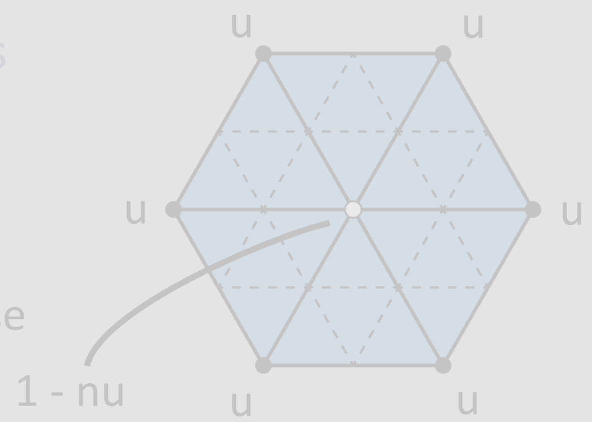
How do we efficiently do Step 1?

Step 2:
Assign new coords



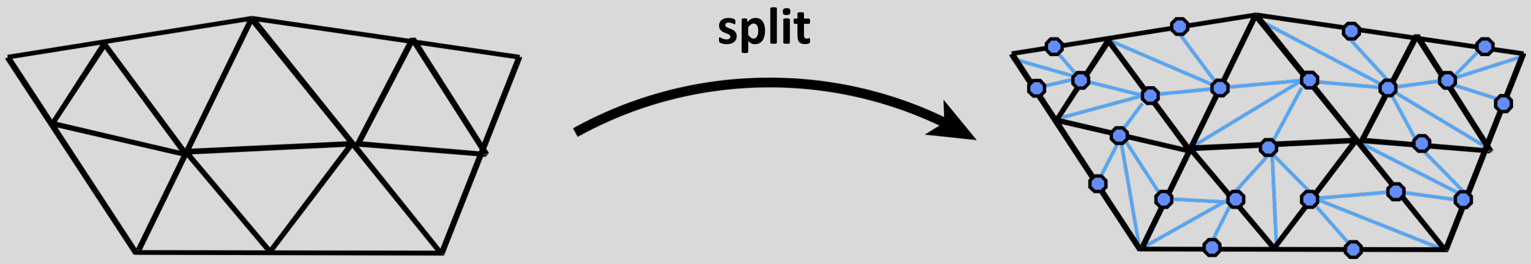
Step 3:
Assign old coords

n - vertex degree
u - 3/16 if n=3
3/(8n) otherwise

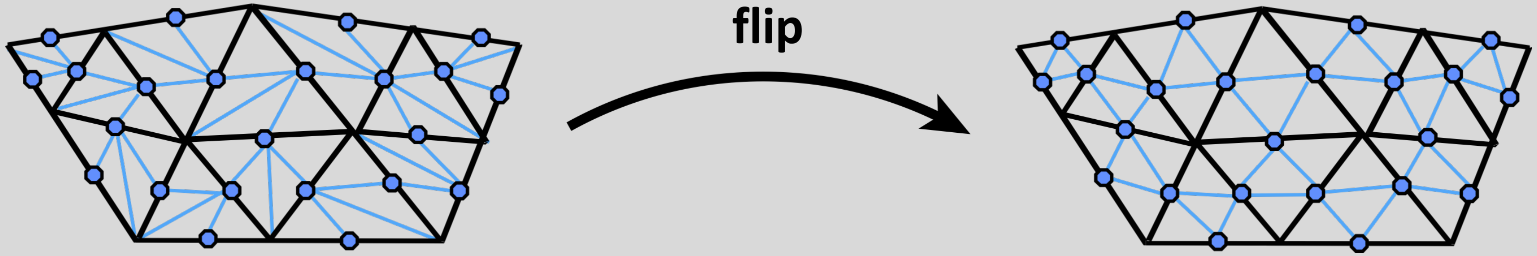


Loop Subdivision Using Local Ops

Step 1:
Split all edges in any order

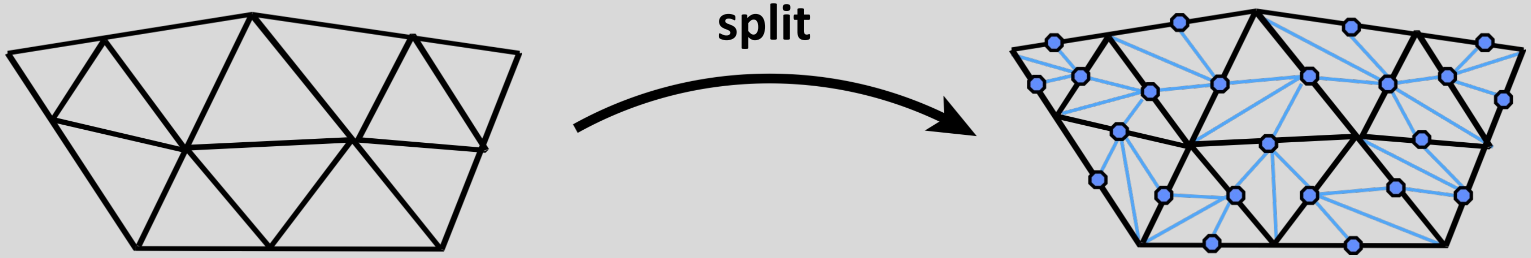


Step 2:
Flip new edges until they touch two new vertices



Loop Subdivision Using Local Ops

Step 1:
Split all edges in any order



Step 2:
Flip new edges until they touch two new vertices

The order we traverse the edges and split them matter!

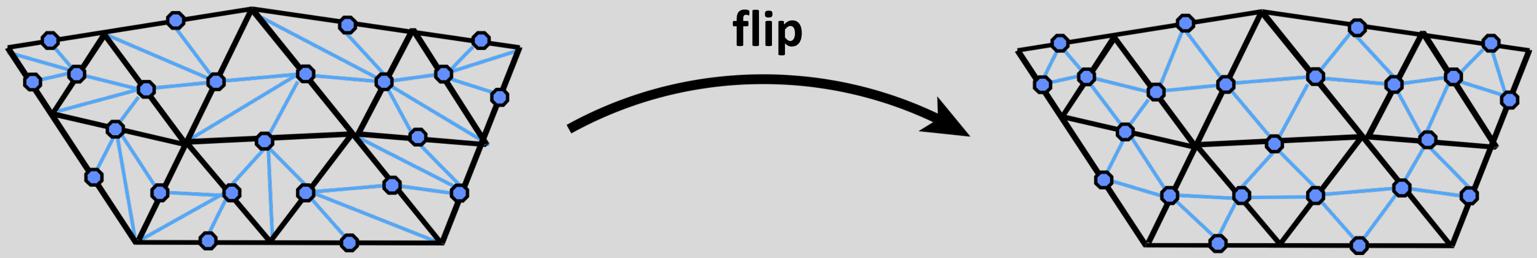
Traversing edges forward and splitting vs traversing them backwards and splitting will yield different meshes

Loop Subdivision Using Local Ops

Step 1:
Split all edges in any order

Flipping new edges until the below criteria is met ensures that any order of splitting edges will still result in the same final mesh

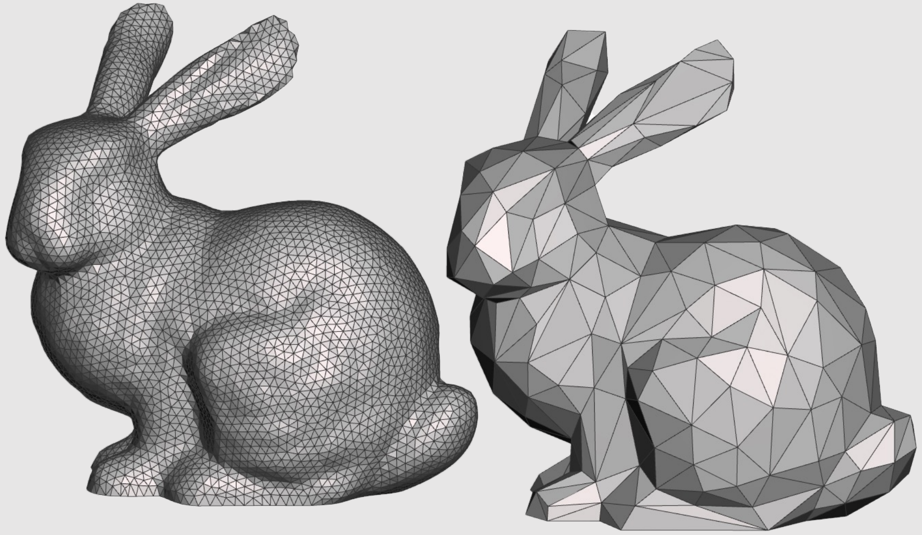
Step 2:
Flip new edges until they touch two new vertices



- Digital Geometric Processing
  - ~~Geometric Subdivision~~
  - Geometric Simplification
  - Geometric Remeshing
  - Geometric Queries

Simplification

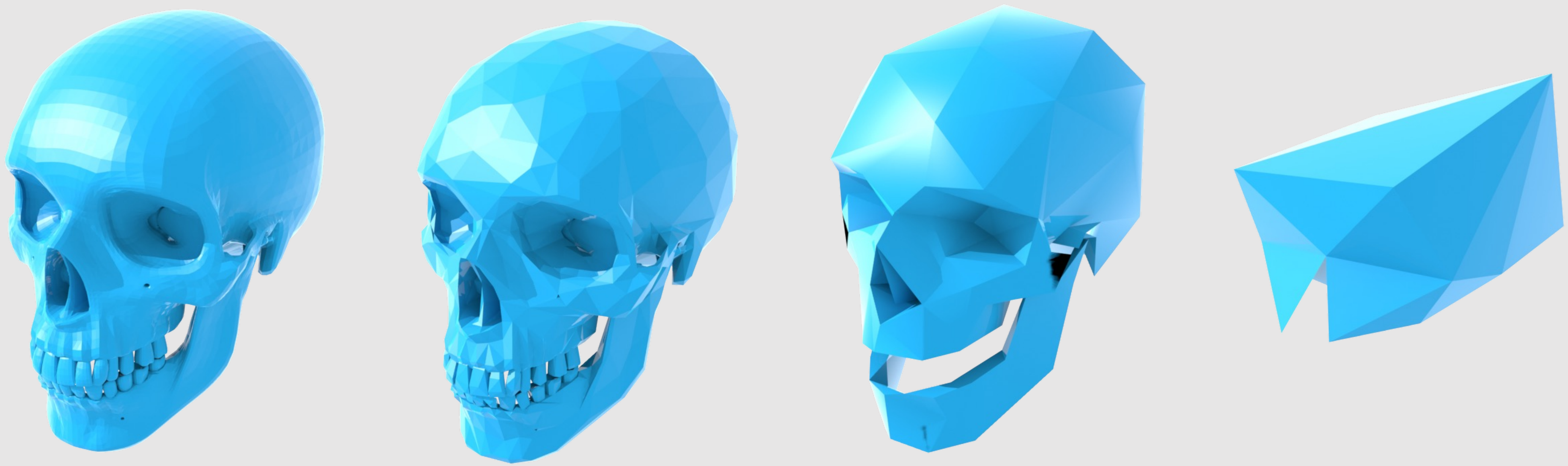
- Simplification is the process of **downsampling** a mesh
  - Less Storage overhead
    - Smaller file sizes
  - Less Processing overhead
    - Less elements to iterate over
  - Larger mesh modifications
    - Instead of moving tens of smaller mesh elements, move one larger mesh element



Simplification Algorithm Basics

- Greedy Algorithm:
  - Assign each edge a cost
  - Collapse edge with least cost
  - Repeat until target number of elements is reached

- Particularly effective cost function: **quadric error metric****



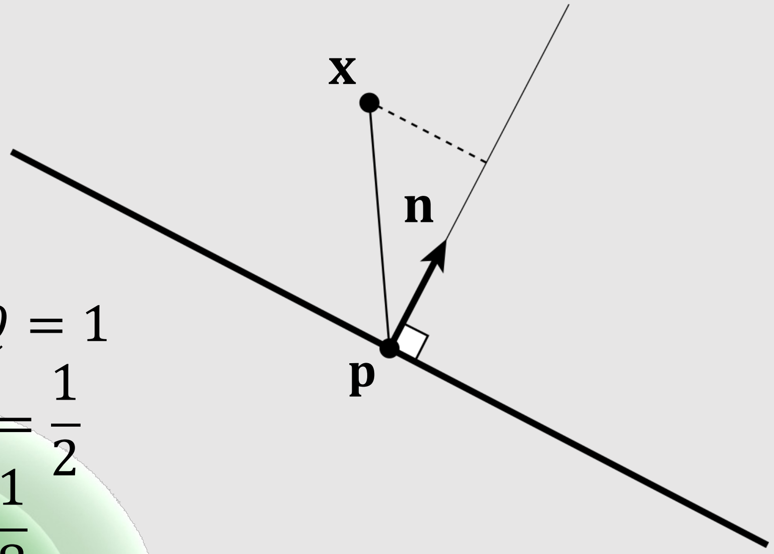
[30,000 triangles] [3,000 triangles] [300 triangles] [30 triangles]

**invented at CMU (Garland & Heckbert 1997)

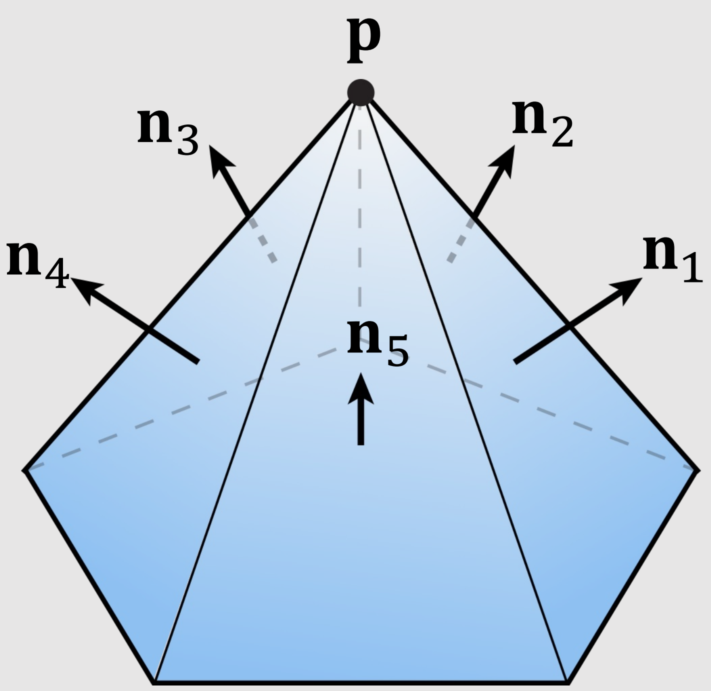
Quadric Error Metric

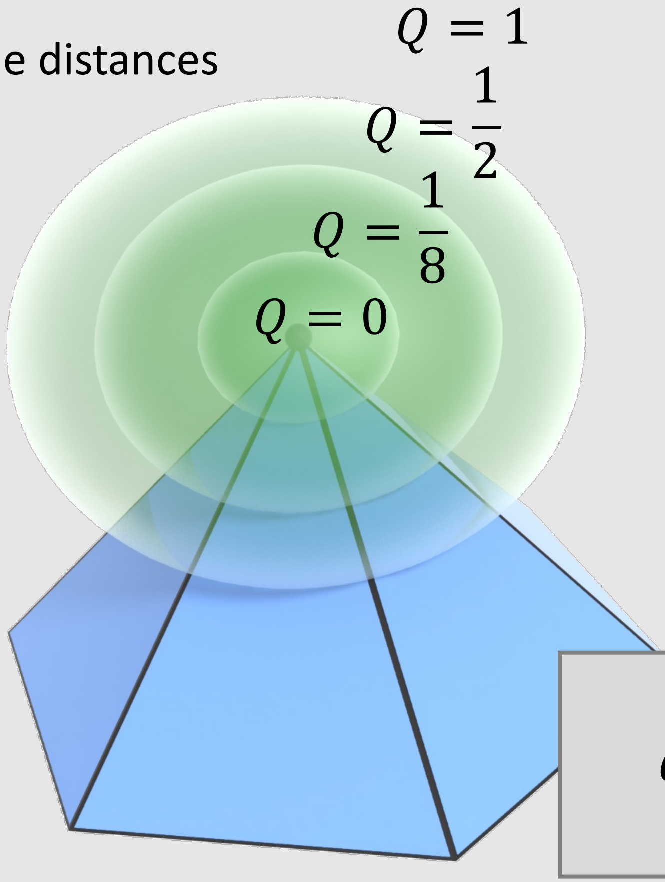
- **Goal:** approximate a point's distance from a collection of triangles
  - **Review:** what is the distance of a point $\mathbf{x}$ from a plane $\mathbf{p}$ with normal $\mathbf{n}$?

$$\mathrm{dist}(\mathbf{x}) = \langle \mathbf{n}, \mathbf{x} \rangle - \langle \mathbf{n}, \mathbf{p} \rangle = \langle \mathbf{n}, \mathbf{x} - \mathbf{p} \rangle$$



- Quadric error is the sum of squared point-to-plane distances





$$Q(\mathbf{x}) := \sum_{i=1}^{k} \langle \mathbf{n}_i, \mathbf{x} - \mathbf{p} \rangle^2$$

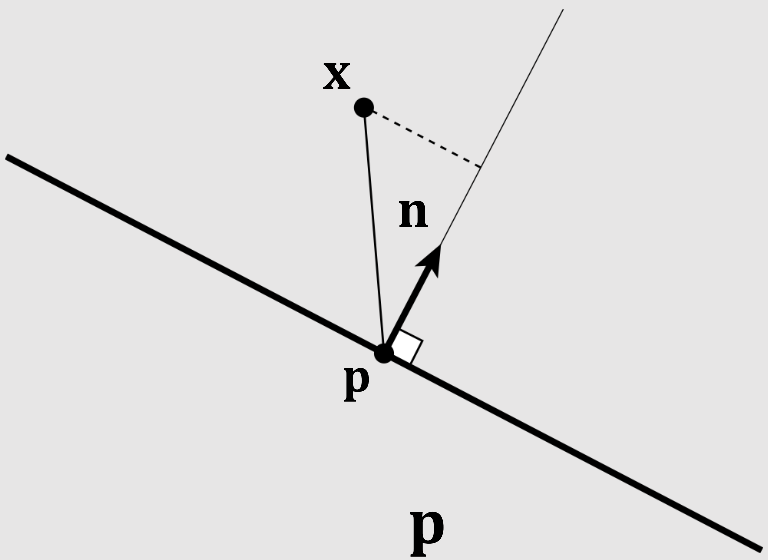
Quadric Error Metric

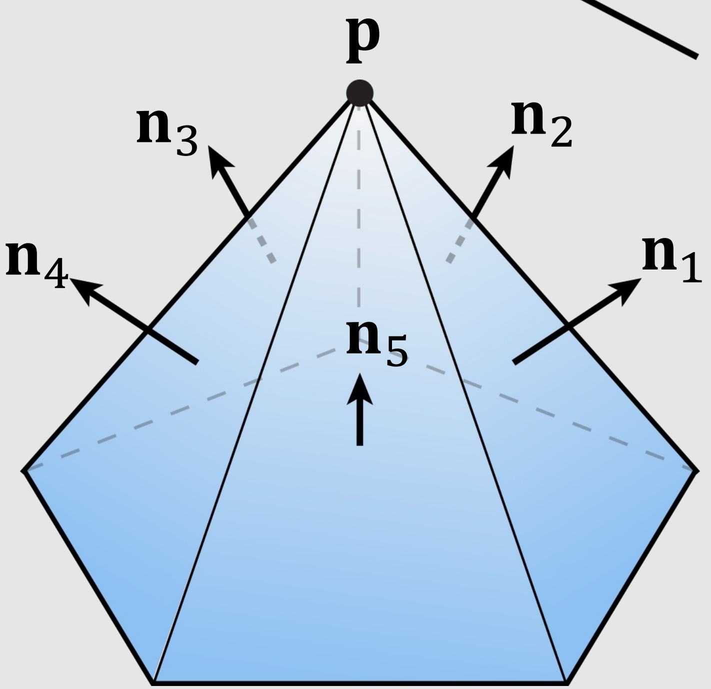
- Given:
  - Query point $\mathbf{x} = (x, y, z)$
  - Normal $\mathbf{n} = (a, b, c)$
  - Offset from origin $e = \langle \mathbf{n}, \mathbf{p} - 0 \rangle = \langle \mathbf{n}, \mathbf{p} \rangle$
  - We want the negative of this value to make a plane equation
    - $d = -e = -\langle \mathbf{n}, \mathbf{p} \rangle$
- We can rewrite in homogeneous coordinates:
  - $\mathbf{u} = (x, y, z, 1)$
  - $\mathbf{v} = (a, b, c, d)$
- Signed distance to plane is then just $\langle \mathbf{u}, \mathbf{v} \rangle = ax + by + cz + d$
  - Note that it is zero in the plane!
- Squared distance is $\langle \mathbf{u}, \mathbf{v} \rangle^2 = \mathbf{u}^\mathsf{T}(\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{u} =: \mathbf{u}^\mathsf{T} K \mathbf{u}$
  - Matrix $K = \mathbf{v}\mathbf{v}^T$ encodes squared distance to plane

$$K = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

$$\mathbf{u}^\mathsf{T} K_1 \mathbf{u} + \mathbf{u}^\mathsf{T} K_2 \mathbf{u} = \mathbf{u}^\mathsf{T}(K_1 + K_2)\mathbf{u}$$

- **Key Idea:** sum of matrices K represents distance to a union of planes





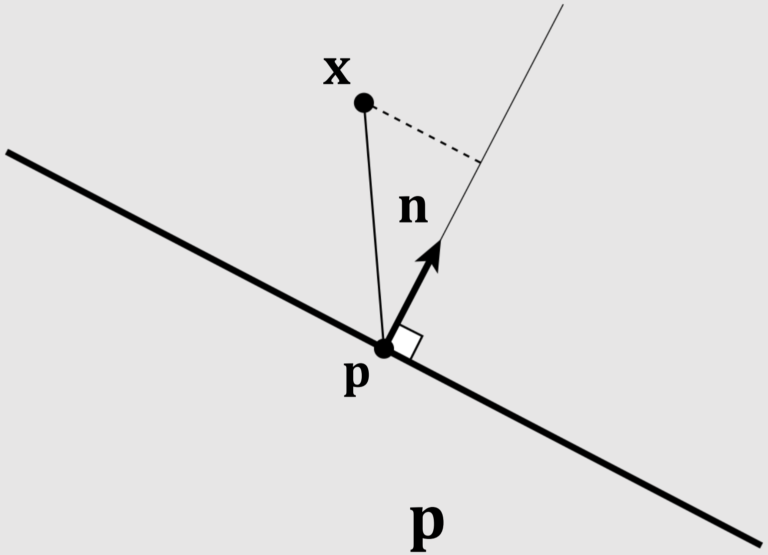
Quadric Error Metric

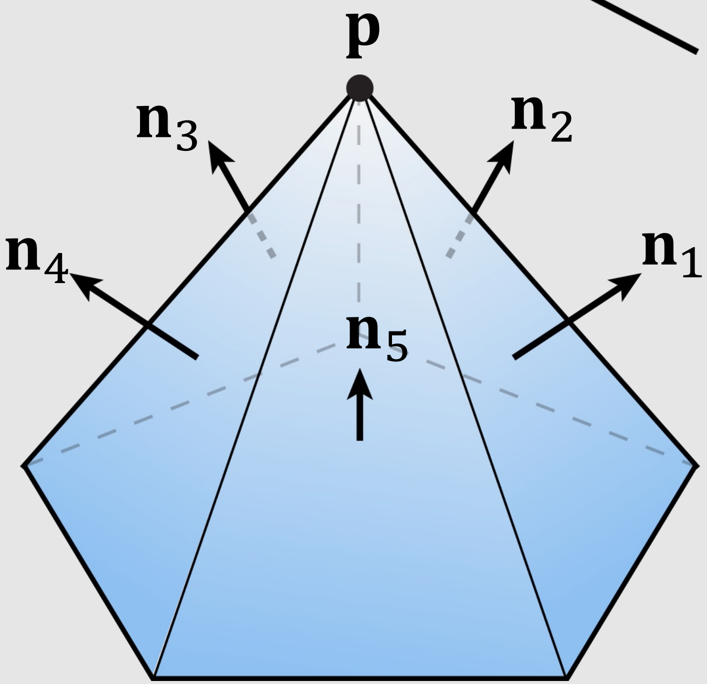
- Given:
  - Query point $\mathbf{x} = (x, y, z)$
  - Normal $\mathbf{n} = (a, b, c)$
  - $d = -\langle \mathbf{n}, \mathbf{p} \rangle$
  - $\mathbf{u} = (x, y, z, 1)$
  - $\mathbf{v} = (a, b, c, d)$
- Signed distance to plane is $\langle \mathbf{u}, \mathbf{v} \rangle = ax + by + cz + d$
- Squared distance is $\langle \mathbf{u}, \mathbf{v} \rangle^2 = \mathbf{u}^\mathsf{T}(\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{u} =: \mathbf{u}^\mathsf{T} K \mathbf{u}$
  - Matrix $K = \mathbf{v}\mathbf{v}^T$ encodes squared distance to plane

$$K = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

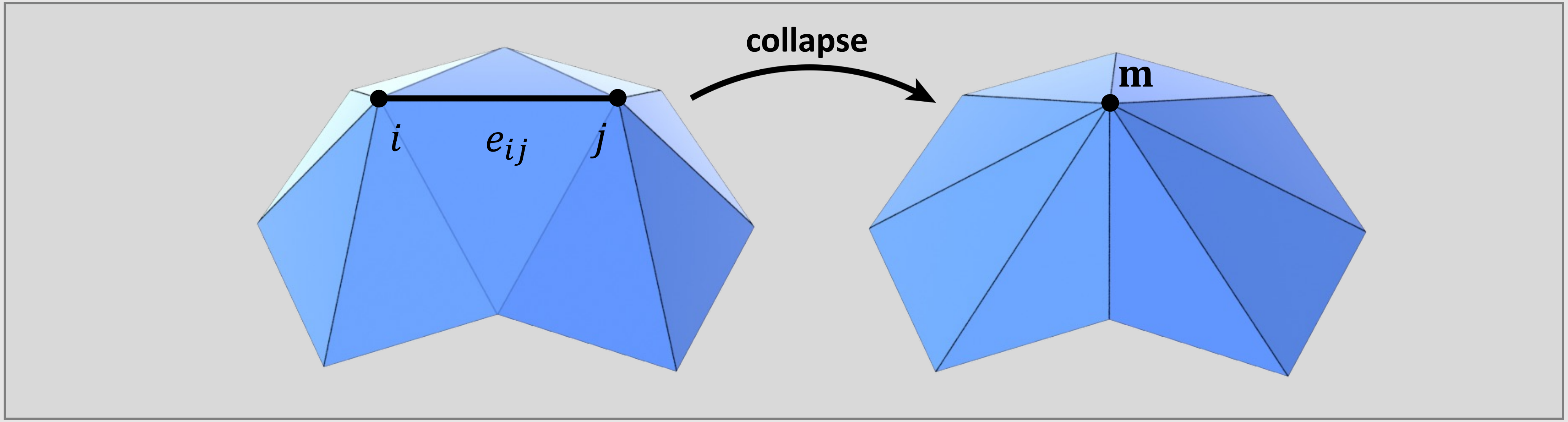
- **Key Idea:** sum of matrices K represents distance to a union of planes

$$\mathbf{u}^\mathsf{T} K_1 \mathbf{u} + \mathbf{u}^\mathsf{T} K_2 \mathbf{u} = \mathbf{u}^\mathsf{T}(K_1 + K_2)\mathbf{u}$$





Quadric Error of Edge Collapse

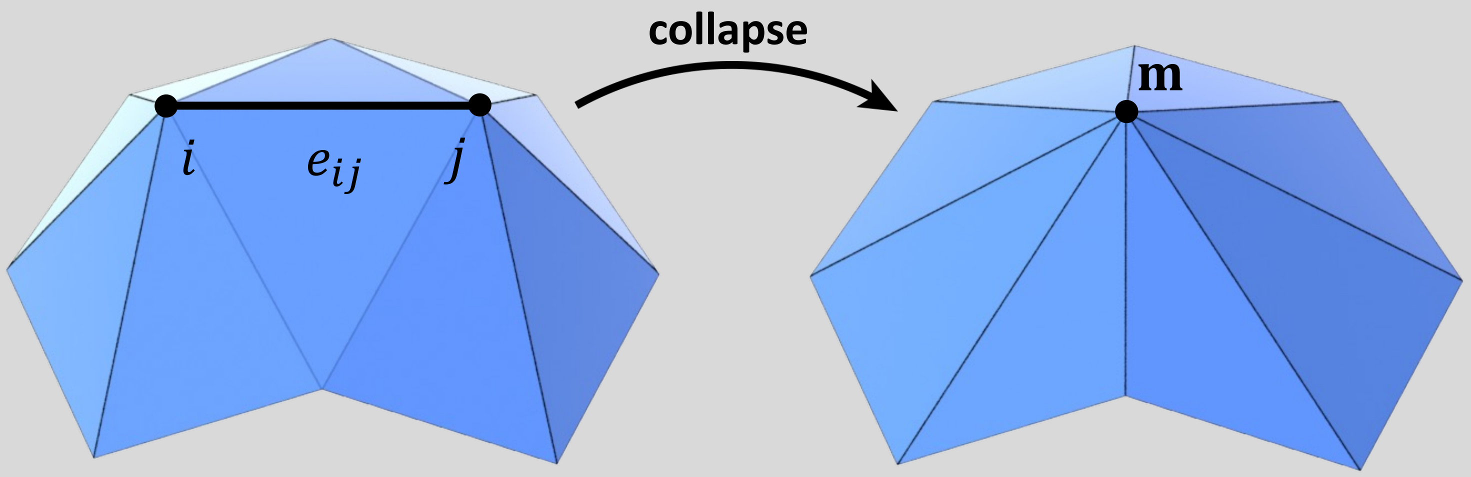


- How much does it cost to collapse an edge e_{ij}?
  - Compute midpoint $\mathbf{m}$, measure error as

$$Q(\mathbf{m}) = \mathbf{m}^{\mathsf{T}}(K_i + K_j)\mathbf{m}$$

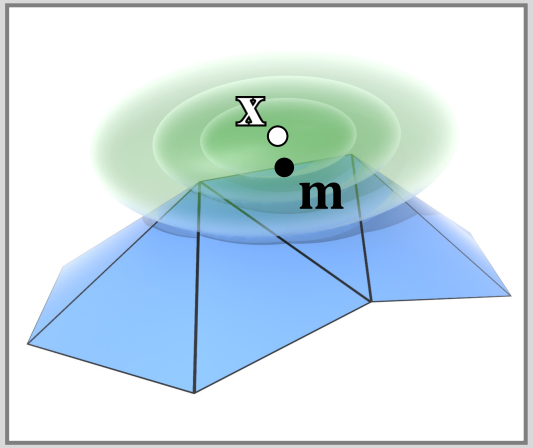
- Error becomes "score" for e_{ij}, determining priority
  - Q: where to put $\mathbf{m}$?

Quadric Error of Edge Collapse



$$Q(\mathbf{m}) = \mathbf{m}^{\mathsf{T}}(K_i + K_j)\mathbf{m}$$

- Find point $\mathbf{x}$ that minimizes error
    - Take derivatives!



How to take a derivative of a function involving matrices?

Minimizing a Quadratic Function

To find the min of a function $f(x)$

$$f(x) = ax^2 + bx + c$$

take derivative $f'(x)$ and set equal to 0

$$f'(x) = 2ax + b = 0$$

$$\boxed{x = -b/2a}$$

same structure

can also write any quadratic function of n variables as a symmetric matrix A
consider the multivariable function

$$f(x, y) = ax^2 + bxy + cy^2 + dx + ey + g$$

we can rewrite it as:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad A = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} d \\ e \end{bmatrix}$$

$$f(x, y) = \mathbf{x}^{\mathsf{T}} A \mathbf{x} + \mathbf{u}^{\mathsf{T}} \mathbf{x} + g$$

take derivative $f'(x)$ and set equal to 0

$$f'(x, y) = 2A\mathbf{x} + \mathbf{u} = 0$$

$$\boxed{\mathbf{x} = -\frac{1}{2} A^{-1} \mathbf{u}}$$

same structure

Positive Definite Quadratic Form

How do we know if our solution minimizes quadratic error?

$$\mathbf{x} = -\frac{1}{2}A^{-1}\mathbf{u}$$

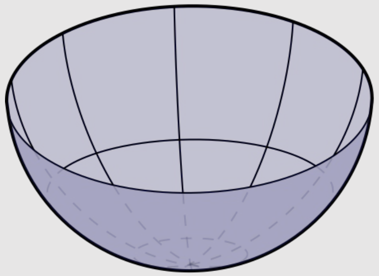
In the 1D case, we minimize the function if

$$xax = ax^2 > 0$$
$$a > 0$$

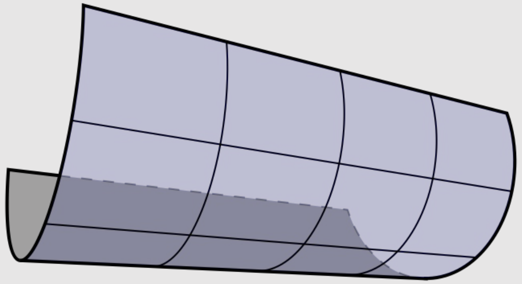
In the ND case, we minimize the function if

$$\mathbf{x}^{\mathsf{T}} A\, \mathbf{x} > 0 \quad \forall\, \mathbf{x}$$

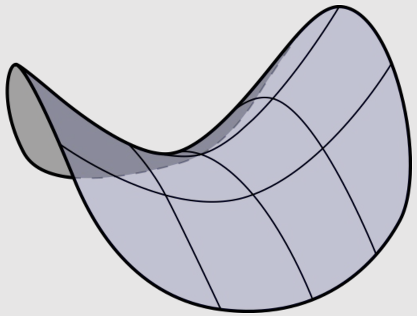
This is known as the function being positive semidefinite



[positive definite]



[positive semidefinite]



[indefinite]

Minimizing Quadric Error

Find "best" point for edge collapse by minimizing quadratic form

$$\min_{\mathbf{u}\in\mathbb{R}^4} \mathbf{u}^T K \mathbf{u}$$

Already know fourth (homogeneous) coordinate for a point is 1
Break up our quadratic function into two pieces

$$\begin{bmatrix} \mathbf{x}^\mathsf{T} & 1 \end{bmatrix} \begin{bmatrix} B & \mathbf{w} \\ \mathbf{w}^\mathsf{T} & d^2 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

$$= \mathbf{x}^\mathsf{T} B \mathbf{x} + 2\mathbf{w}^\mathsf{T}\mathbf{x} + d^2$$

Can minimize as before

$$2B\mathbf{x} + 2\mathbf{w} = 0$$
$$\mathbf{x} = -B^{-1}\mathbf{w}$$

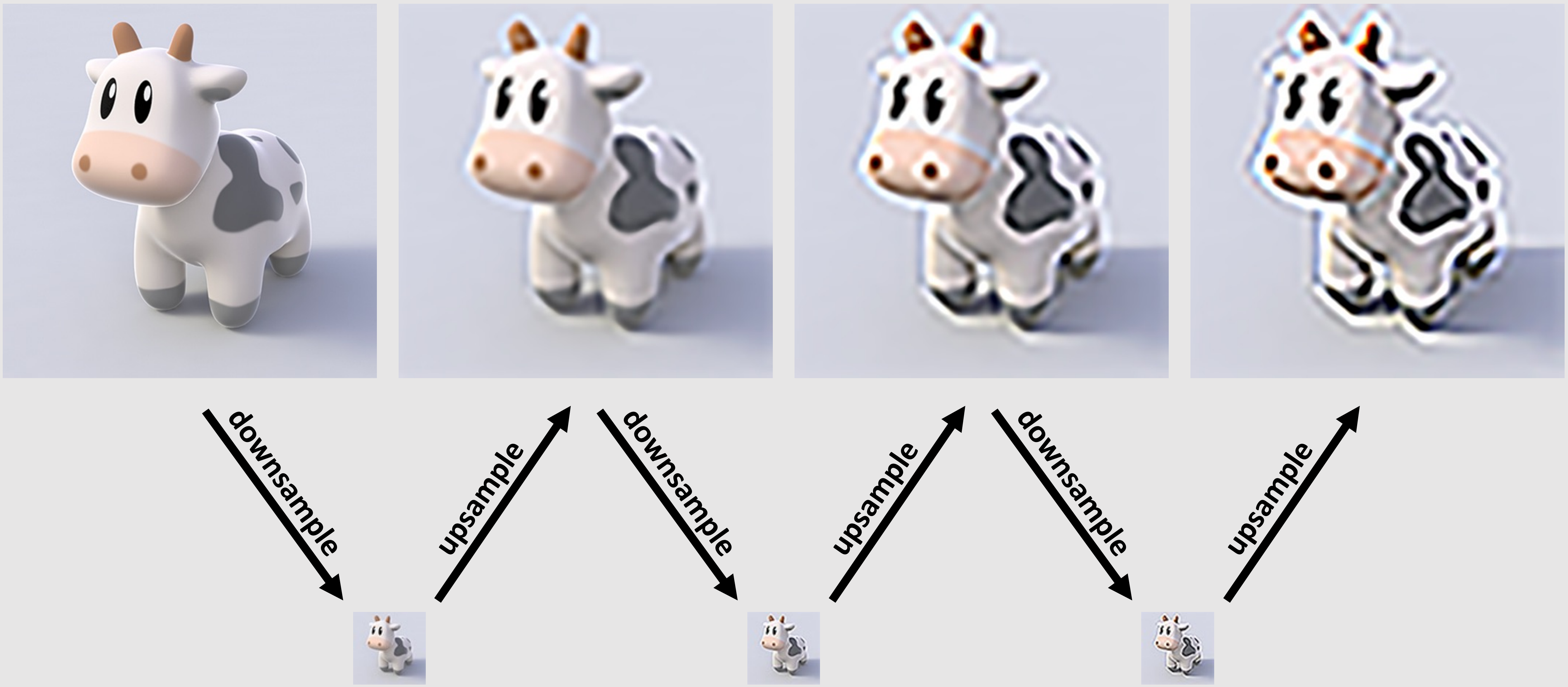
Quadric Error Simplification Algorithm

```
// compute K for each face
for(v : vertices) {
        for(f : faces) {
                Vec4 ve(N, d);
                f->K = outer(ve, ve);
        }
}

// compute K for each vertex
for(v : vertices)
        for(f : v->faces())
                v->K += f->K;

// compute K for each edge
// place into priority queue
PriorityQueue pq;
for(e : edge) {
        for(v : e->vertices())
                e->K += v->K;
        pq.push(e->K, e);
}
```

```
// iterate until mesh is a target size
while(faces.length() > target_size) {

        // collapse edge with smallest cost
        e = pq.pop();
        K = e->K;
        v = collapse(e);

        // position new vertex to optimal pos
        v->pos = -B.inv() * w

        // update K for vertex
        // update K for edges touching vertex
        v->K = K;
        for(e2 : v->edges()) {
                e2->K = 0
                for(v2 : e2->vertices())
                        e2->K += v2->K;
        }
}
```

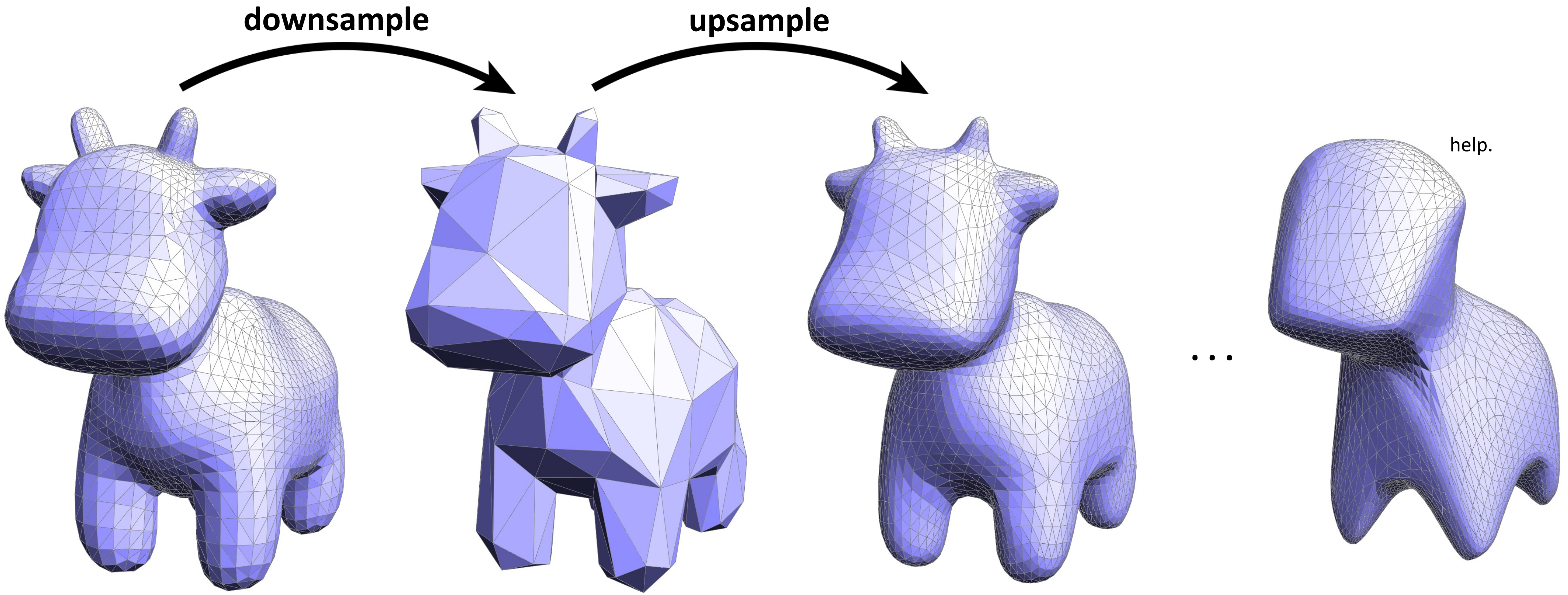
Is simplification the inverse operation of subdivision?

Dangers of Resampling



Repeatedly resampling an image degrades signal quality!

Dangers of Resampling



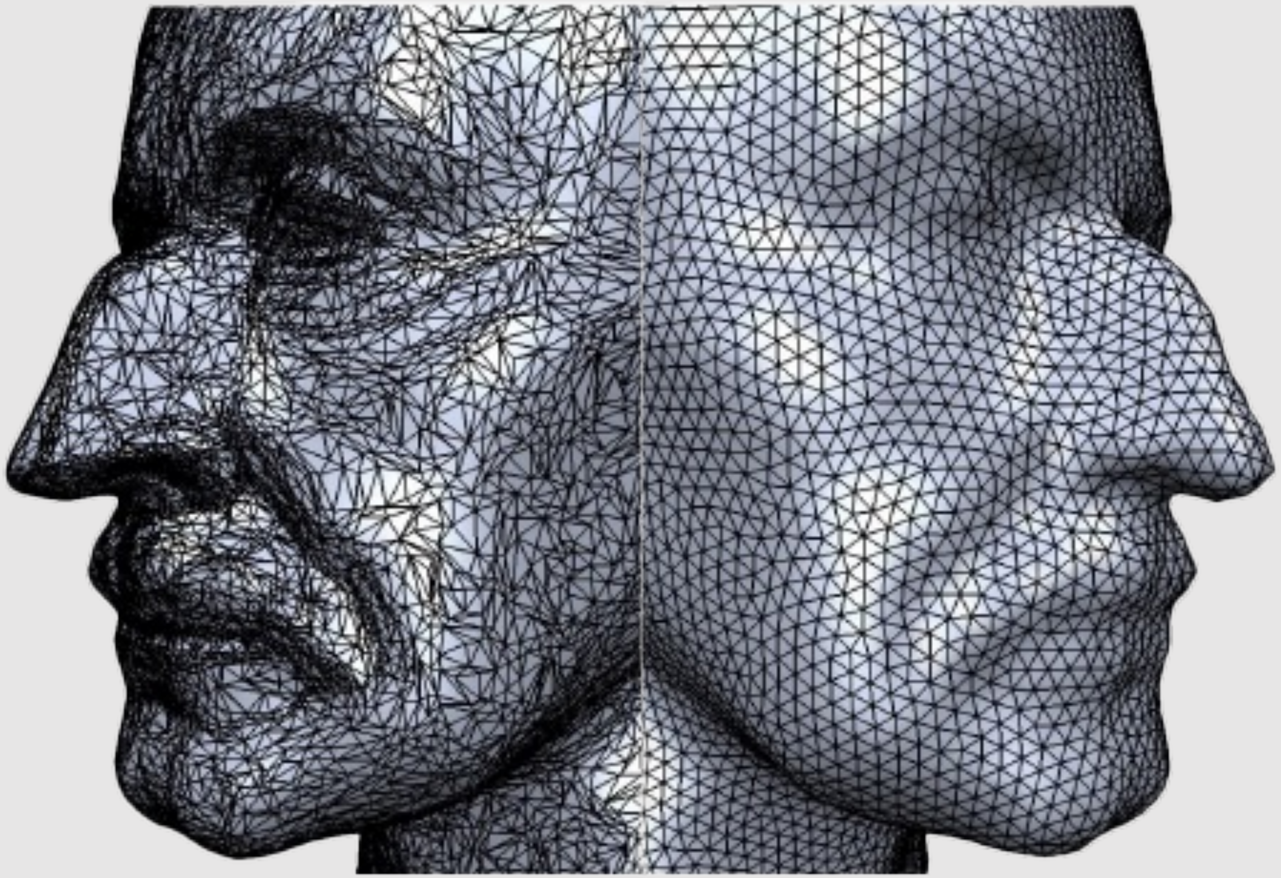
Repeatedly resampling a mesh also degrades signal quality!

- Digital Geometric Processing
    - ~~Geometric Subdivision~~
    - ~~Geometric Simplification~~
    - Geometric Remeshing
    - Geometric Queries

Isotropic Remeshing

- **Isotropic:** same value when measured in any direction
- **Remeshing:** a change in the mesh
  - **Goal:** change the mesh to make triangles more uniform shape and size

- Helps achieve good mesh properties:
  - Good approximation of original shape
  - Vertex degrees close to 6
  - Angles close to 60deg
  - Delaunay triangles

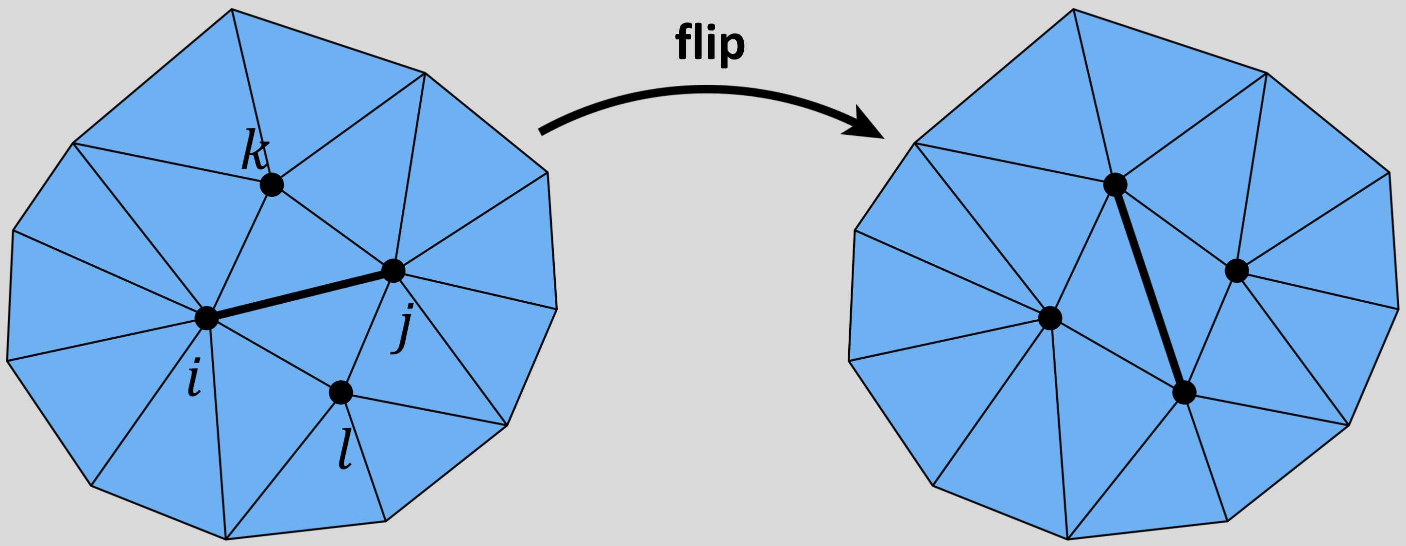


Improving Degree

Vertices with degree 6 makes triangles more regular

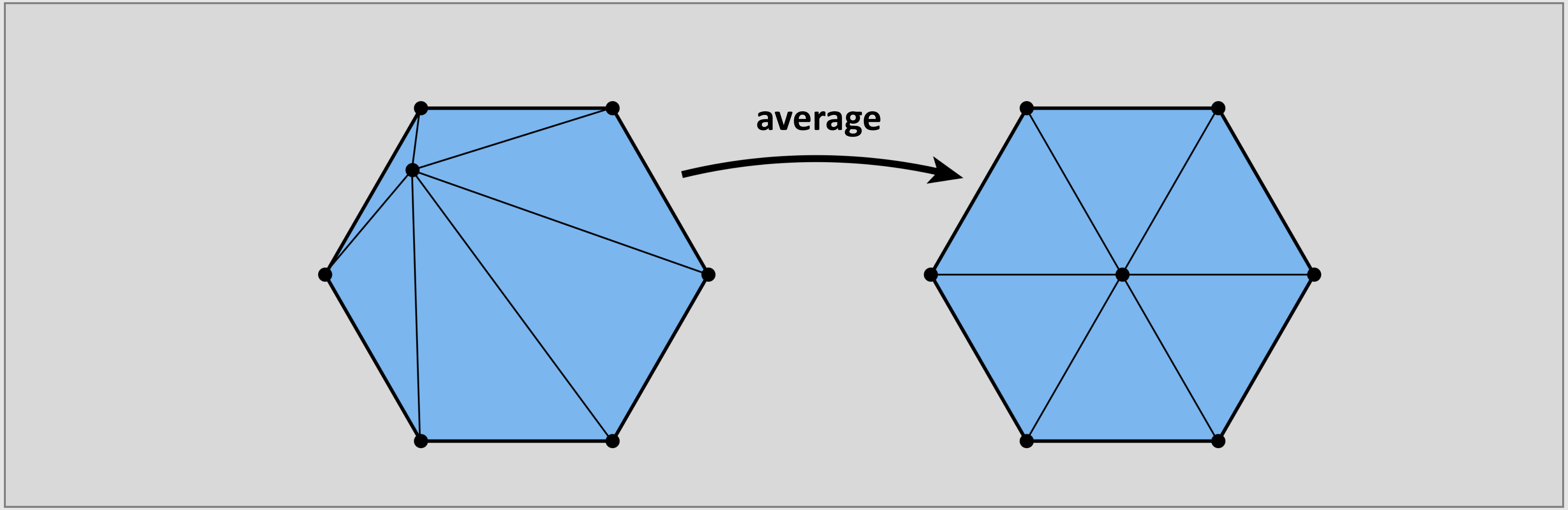
Deviation function: $|d_i - 6| + |d_j - 6| + |d_k - 6| + |d_l - 6|$

If flipping an edge reduces deviation function, flip edge



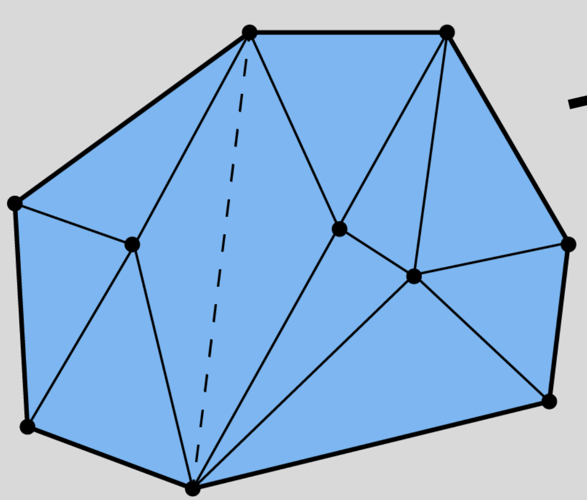
Improving Vertex Positioning

Center vertices to make triangles more even in size



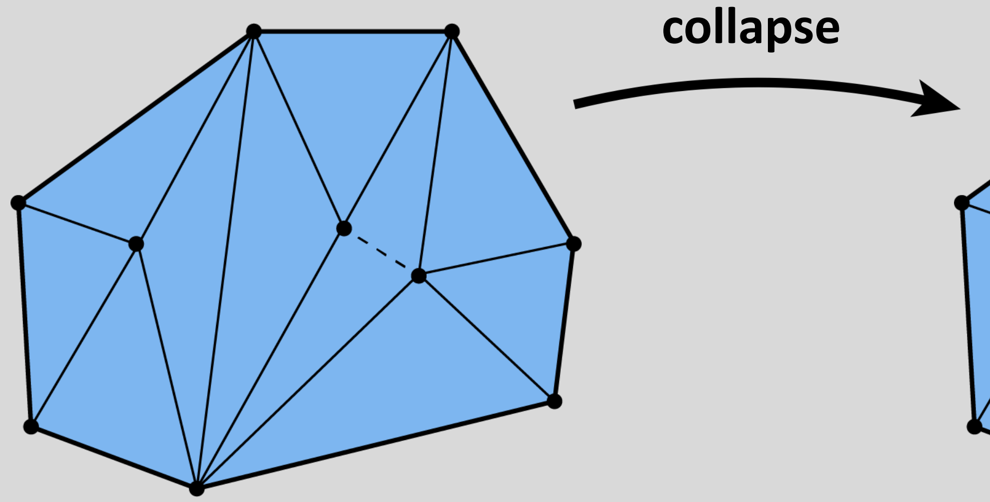
Improving Edge Length

If an edge is longer than (4/3 * mean) length, split it



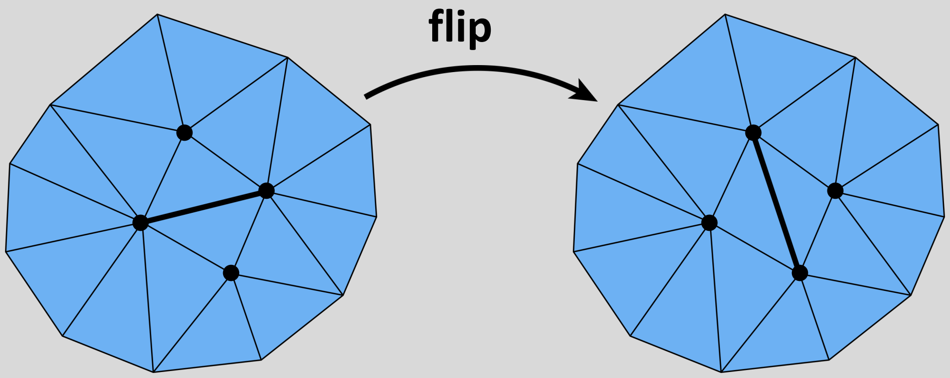
Improving Edge Length

If an edge is shorter than (4/5 * mean) length, collapse it

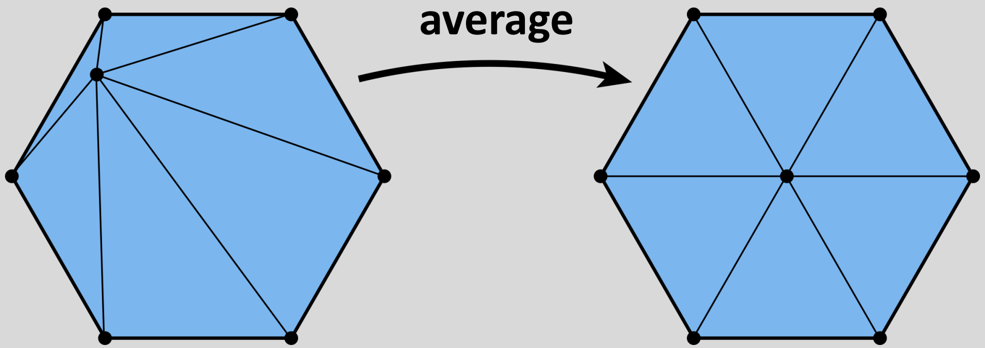


Isotropic Remeshing

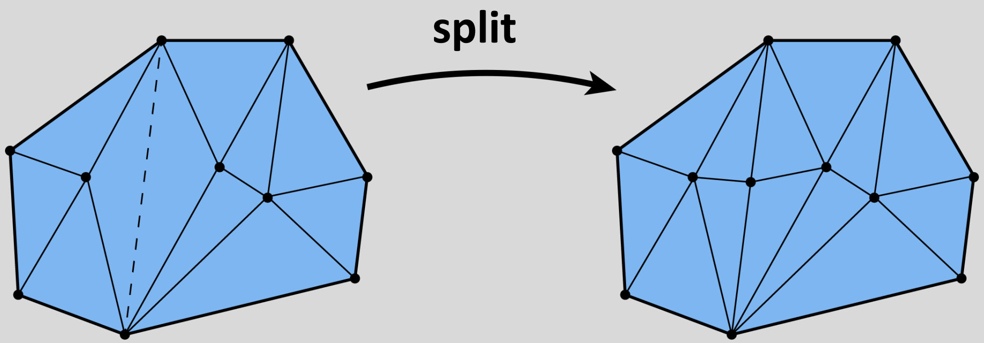
Step 1:



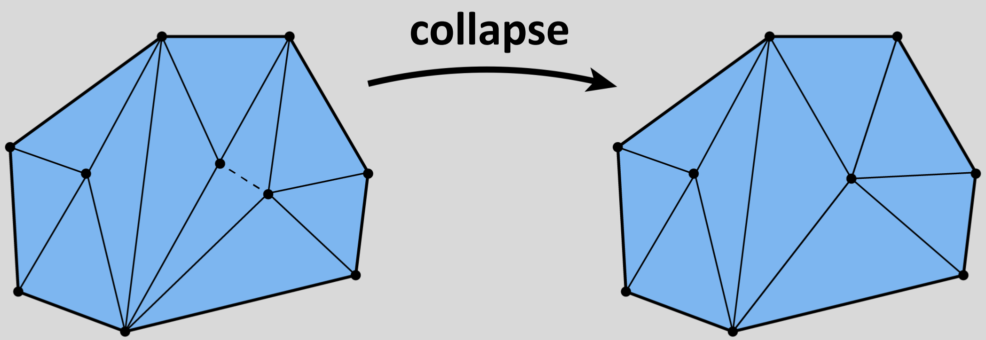
Step 2:



Step 3:



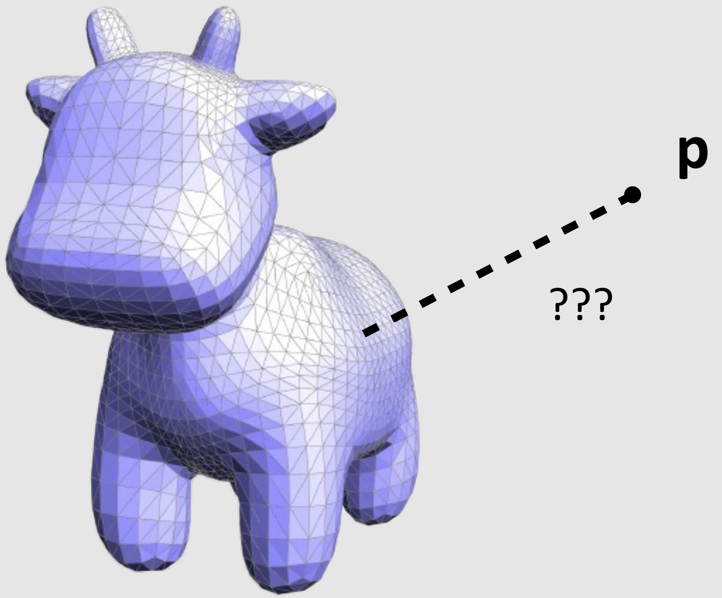
Step 4:



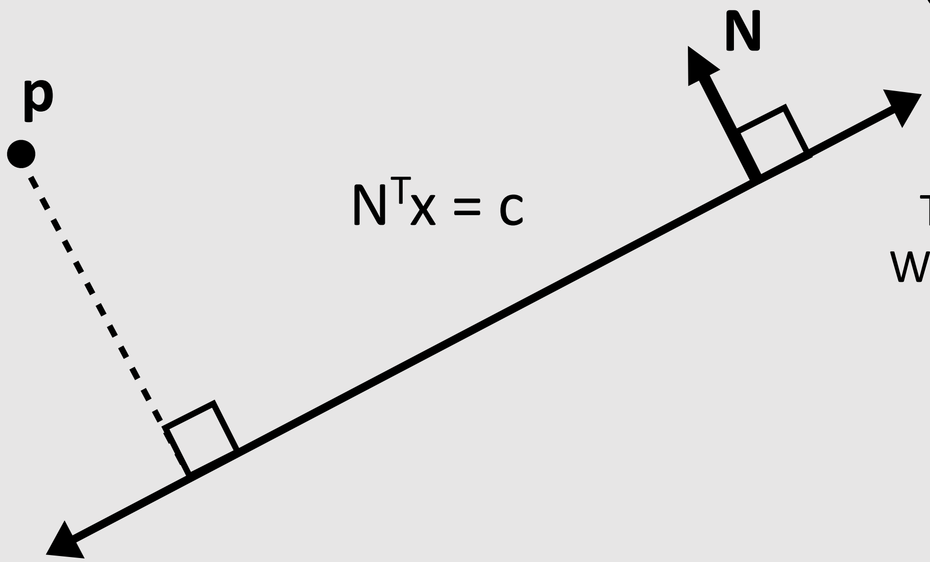
- Digital Geometric Processing
    - ~~Geometric Subdivision~~
    - ~~Geometric Simplification~~
    - ~~Geometric Remeshing~~
    - Geometric Queries

Closest Point Queries

- **Problem:** given a point, in how do we find the closest point on a given surface?
- Several use cases:
  - Ray/mesh intersection in pathtracing
  - Kinematics/animation
  - GUI/user selection
    - When I click on a mesh, what point am I actually clicking on?



Closest Point on a Line



To find the closest point to **p** along $N^Tx = c$
We can have **p** travel along **N** for some time t

$$N^T(p + tN) = c$$

Multiplying the terms out

$$N^Tp + tN^TN = c$$

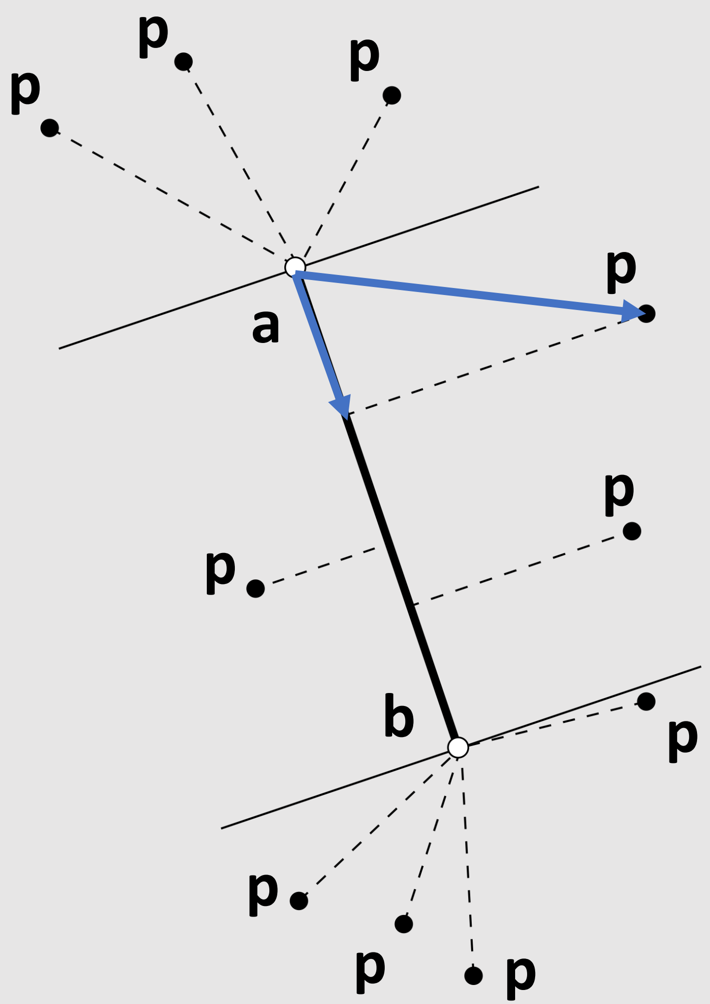
The unit norm multiplied by itself is 1
Solve for t

$$t = c - N^Tp$$

Propagate **p** along **N** for time t

$$p + tN$$
$$p + (c - N^Tp)N$$

Closest Point on a Line Segment



Compute the vector **p** from the line base **a** along the line

$$\langle \mathbf{p} - \mathbf{a}, \mathbf{b} - \mathbf{a} \rangle$$

Normalize to get a time

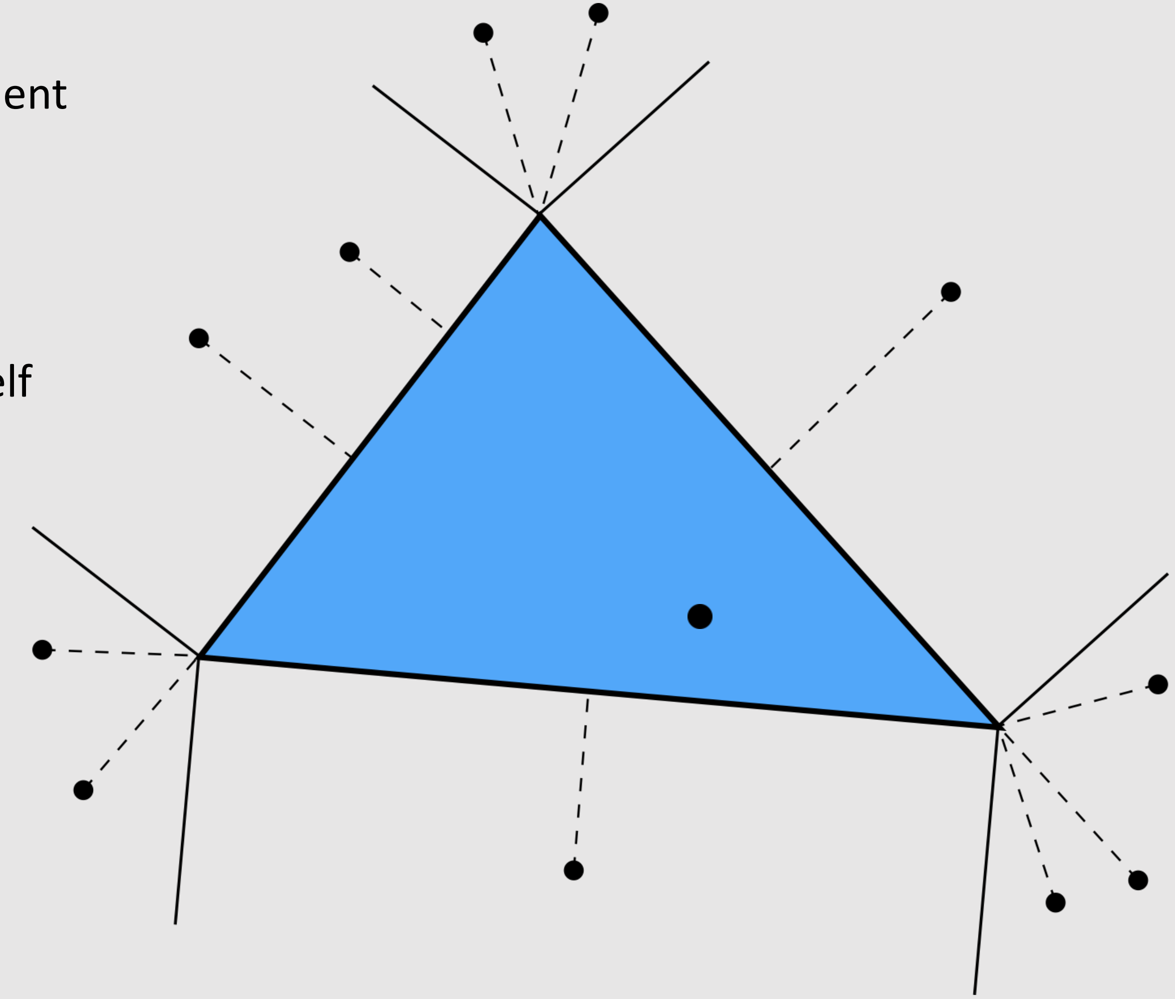
$$t = \frac{\langle \mathbf{p} - \mathbf{a}, \mathbf{b} - \mathbf{a} \rangle}{\langle \mathbf{b} - \mathbf{a}, \mathbf{b} - \mathbf{a} \rangle}$$

Clip time to range [0,1]and interpolate

$$\boldsymbol{a} + (\mathbf{b} - \mathbf{a})t$$

Closest Point on a 2D Triangle

- Easy! Just compute closest point to each line segment
  - For each point, compute distance
  - Point with smallest distance wins
- What if the point is inside the triangle?
  - Even easier! The closest point is the point itself
  - Recall point-in-triangle tests



Closest Point on a 3D Triangle

- **Method #1:** Projection**
  - Construct a plane that passes through the triangle
    - Can be done using cross product of edges
  - Project the point to the closest point on the plane
    - Same expression as with a line: $p + (c - N^T p)N$
    - Check if point is in triangle using half-plane test
  - Else, compute distance from each line segment in 3D
    - Same expression as with a 2D line segment

- **Method #2:** Rotation**
  - Translate point + triangle so that triangle vertex v1 is at the origin
  - Rotate point + triangle so that triangle vertex v2 sits on the z-axis
  - Rotate point + triangle so that triangle vertex v3 sits on the yz-axis
  - Disregard x-coordinate of point
    - Problem reduces to closest point on 2D triangle

**https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.4264&rep=rep1&type=pdf

Closest Point on a 3D Triangle Mesh

- Conceptually easy!
  - Loop over every triangle
  - Compute closest point to current triangle
  - Keep track of globally closest point

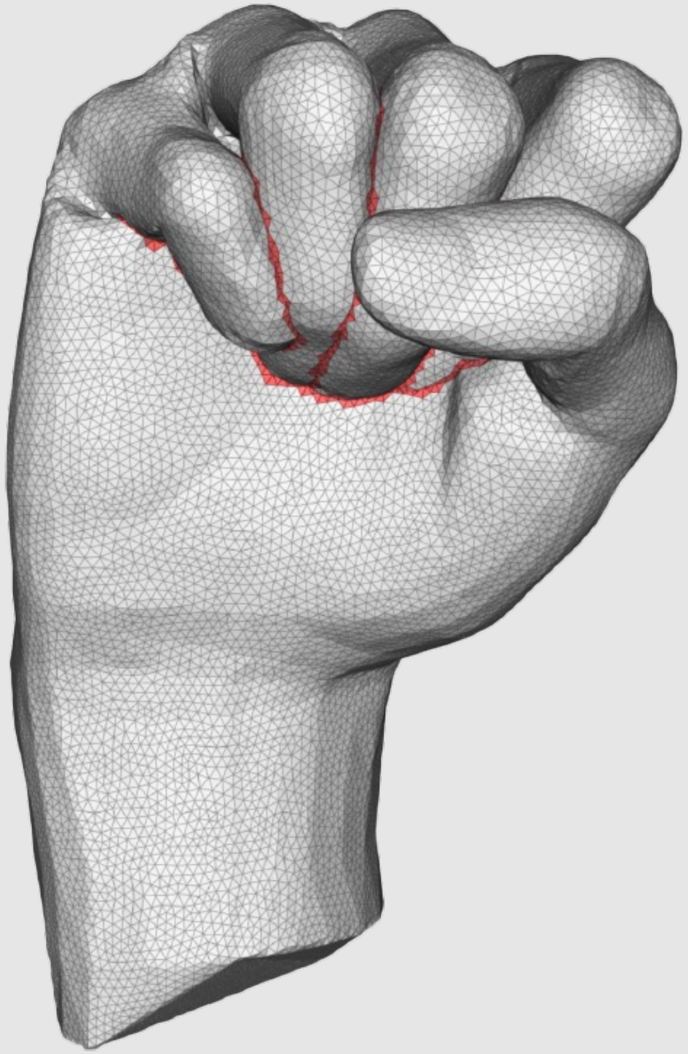
- Not practical in real world
  - Meshes have billions of triangles
  - Programs make thousands of geometric queries a second

- Will look at better solutions next time

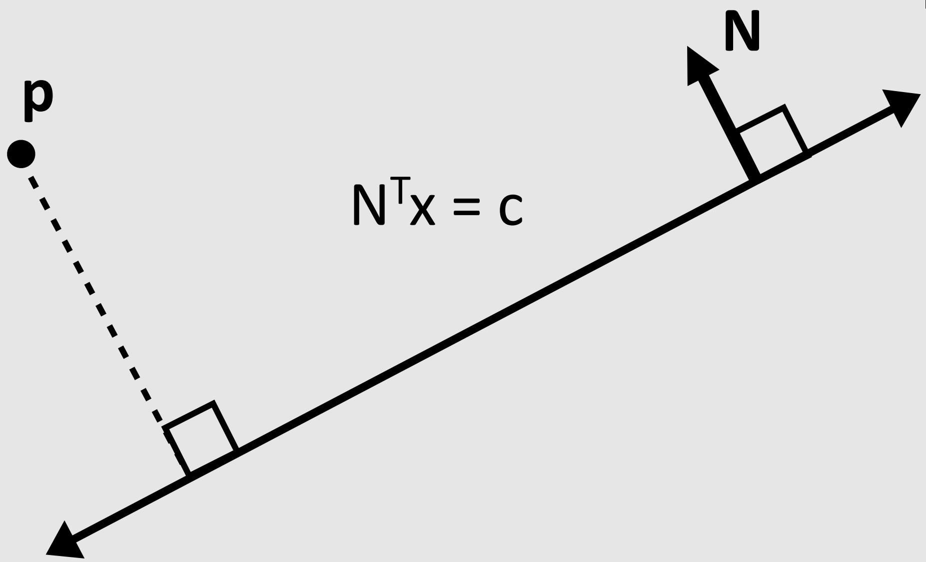
Mesh-Mesh Intersections

- Sometimes when editing geometry, a mesh will intersect with itself
- Likewise, sometimes when animating geometry, meshes will collide

- How do we check for/prevent collisions?



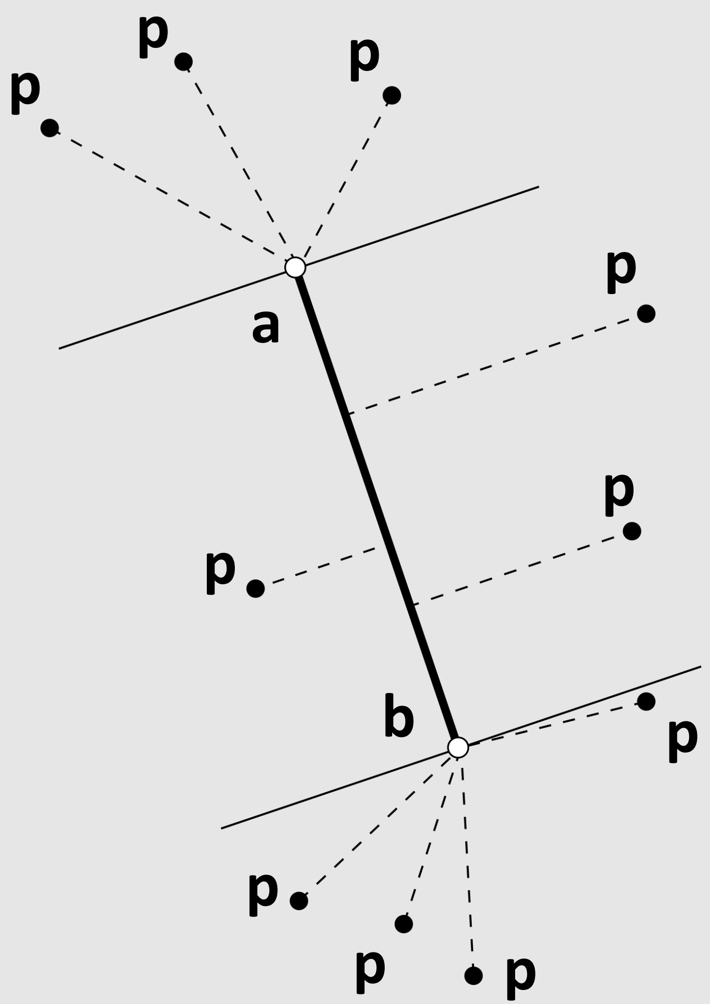
Point-Line Intersection



Just plug point in

$$N^T p = c?$$

Point-Line Segment Intersection



Check if adding distances equals net distance**

$$dist(a,p) + dist(p,b) = dist(a,b)$$

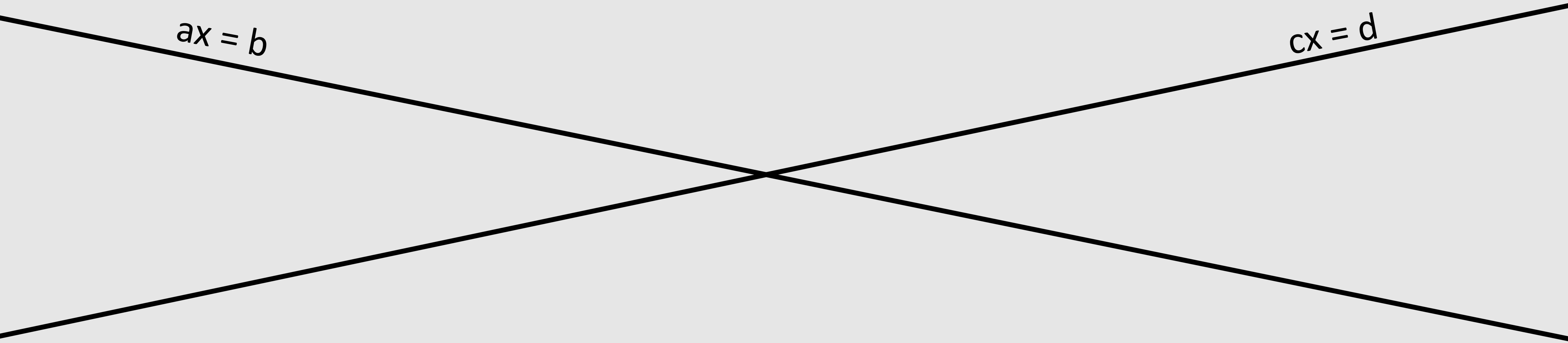
**Potential numeric stability issues

Line-Line Intersection

Two equations, two unknowns

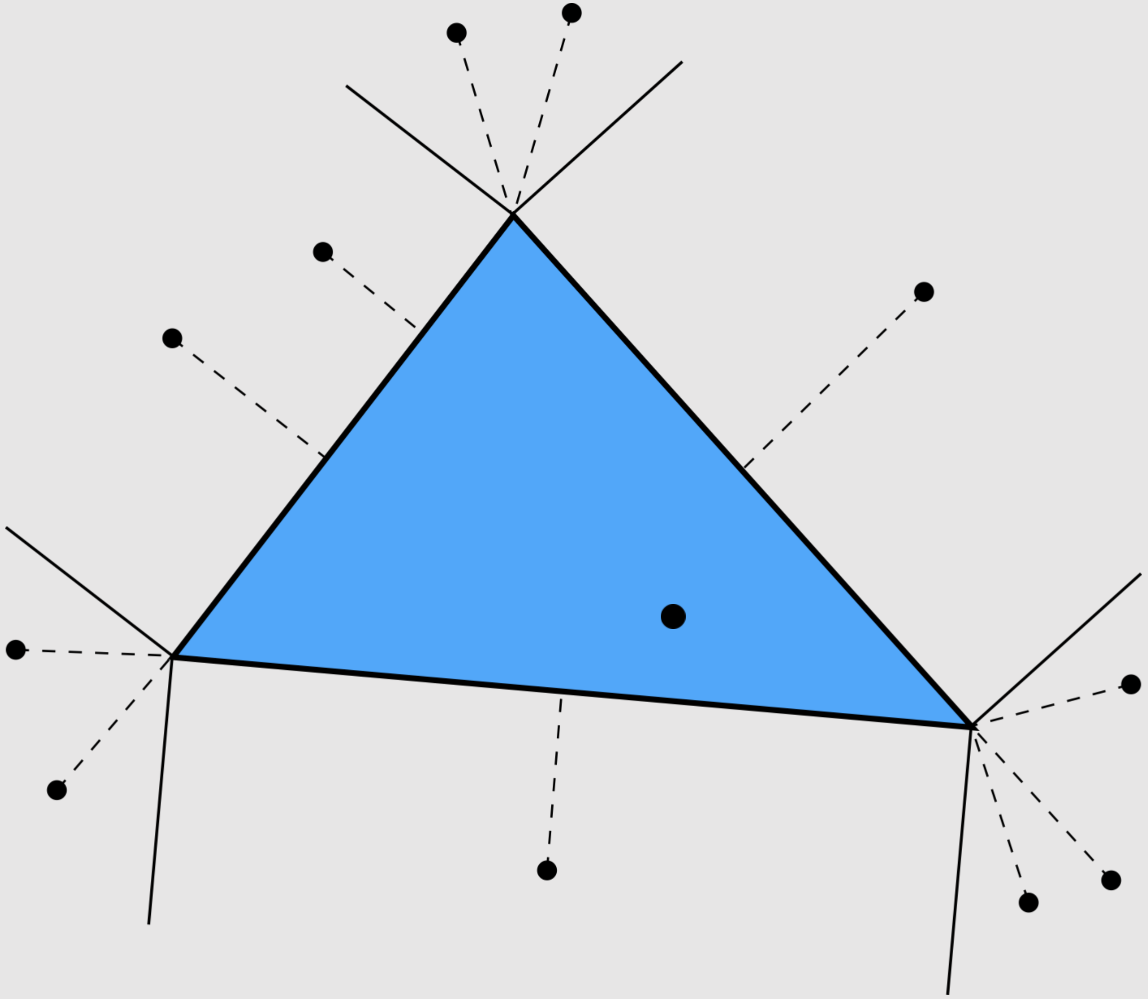
Solve a linear system

$$\begin{bmatrix} a_1 & a_2 \\ c_1 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$



Point-Triangle Intersection

You know this :)



Special Topics in A2: Geometric Representations

- Marching Cubes
- Signed Distance Fields
- NERFs

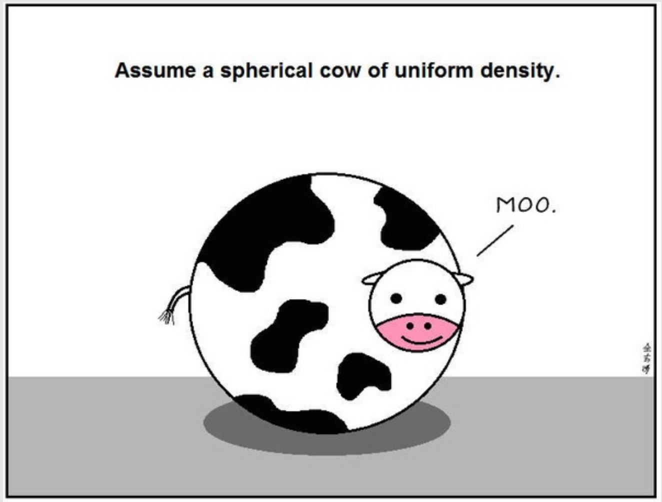
Explicit vs Implicit

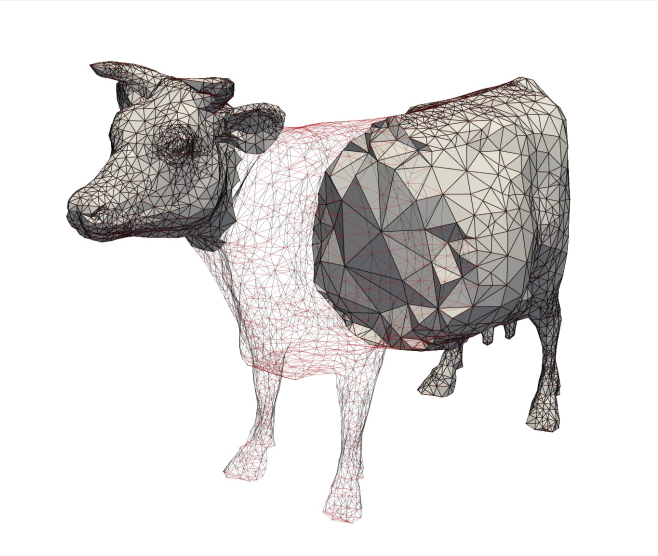
- Not one ideal geometry

- **Explicit:**
  - [+] finding any point on the surface
  - [-] finding if a given point lies on the surface

- **Implicit:**
  - [-] finding any point on the surface
  - [+] finding if a given point lies on the surface

- Pick the geometry best for the task at hand!





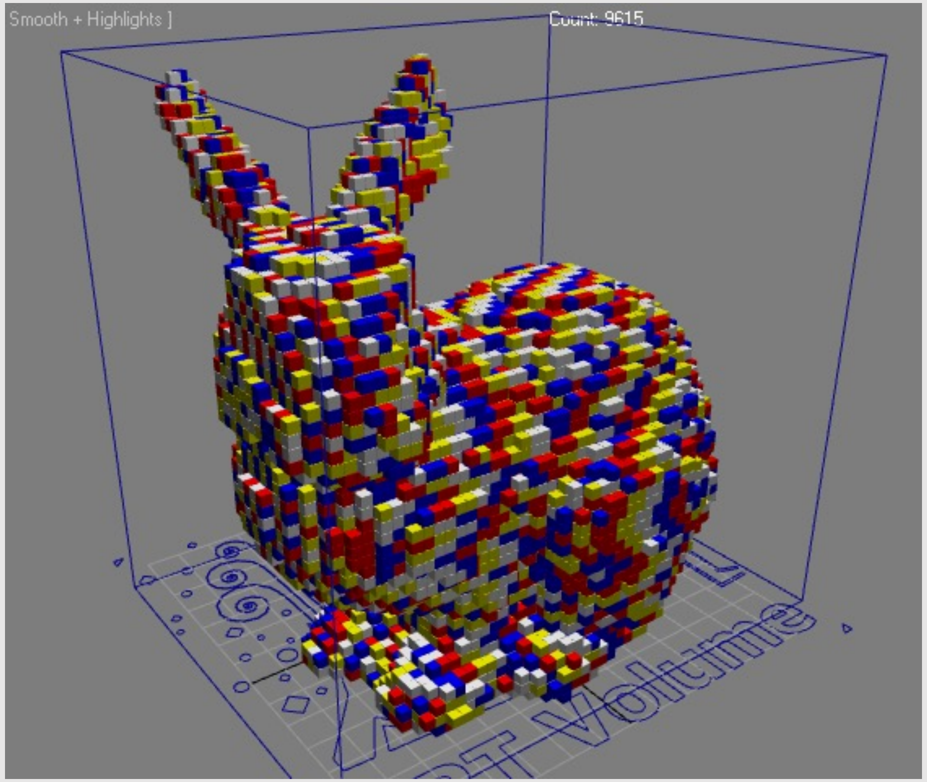
How do we convert implicit geometry to explicit geometry?

Voxel Grid

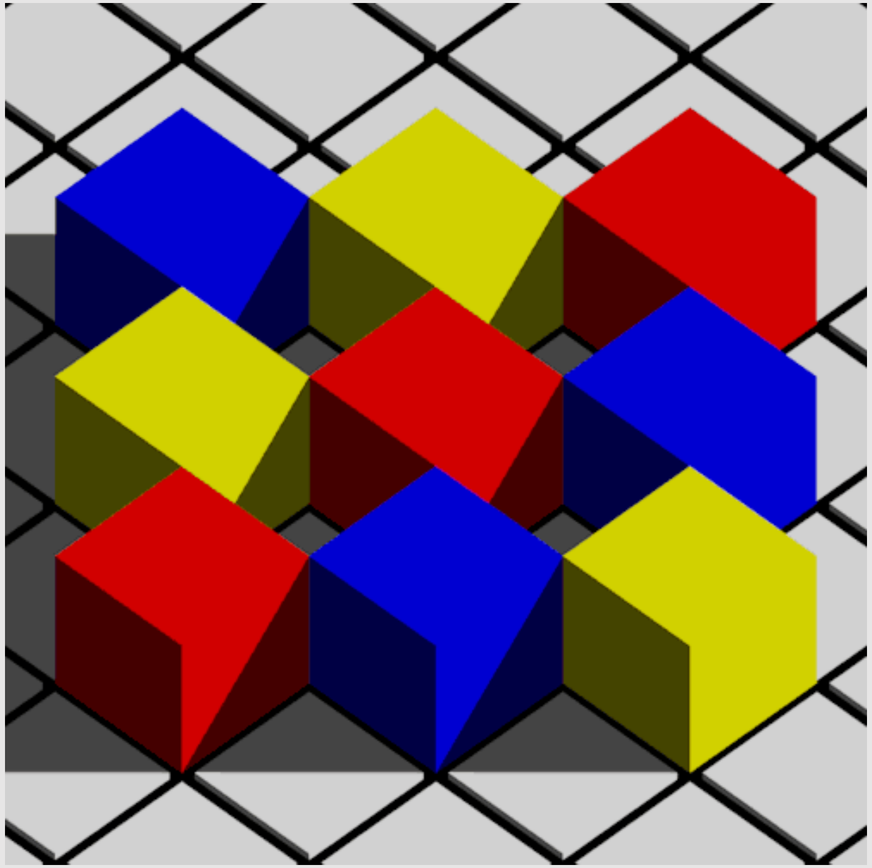
- **Idea:** for an implicit function $f(x, y, z)$, sample points uniformly along the function's domain
  - Plot points where $f(x, y, z) = 0$
  - Results in point cloud

- **Issue:** how many samples to take
  - More samples lead to higher precision, but are more expensive to compute

- **Issue:** does not include any info on connectivity
  - Difficult to interpolate data



Marching Cubes

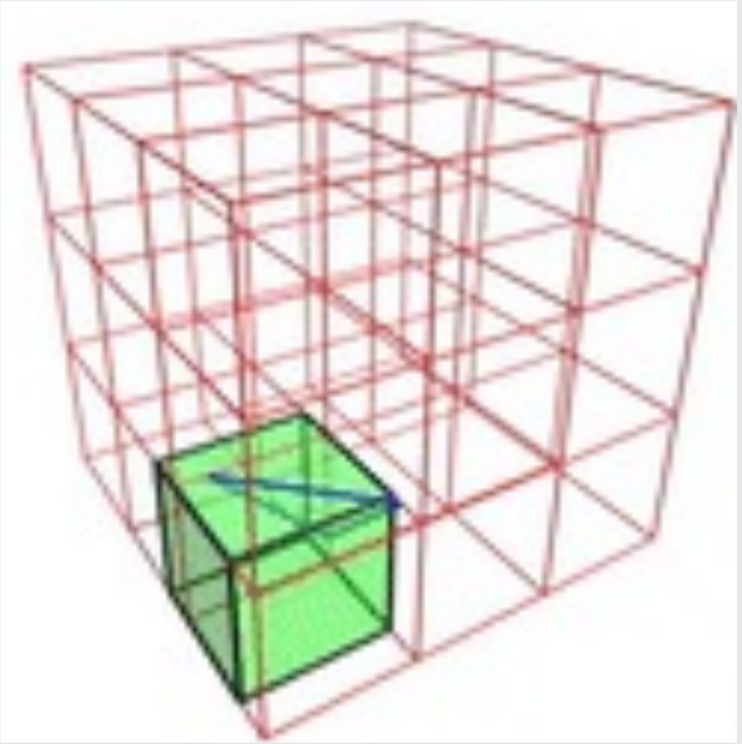

This is not the marching cubes algorithm.
This is literally cubes marching.

- Marching cubes is an algorithm for converting implicit geometry to explicit
  - Adds both positional (vertices) and connectivity (edges)

Marching Cubes

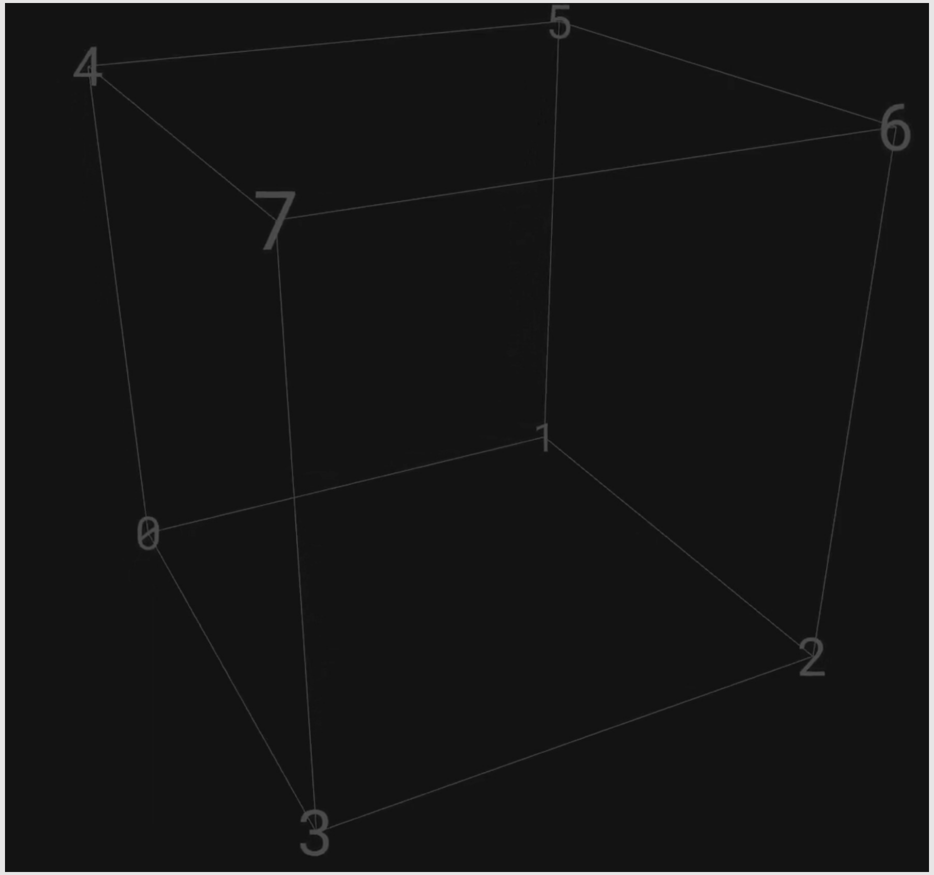
- **Idea:** march a cube though the scene, checking if each of the vertices in the cube lie inside or outside the implicit function $f(x, y, z)$
  - 8 vertices, 8 checks
  - Can encode as an 8-bit number
  - Generate geometry that makes sure inside vertices are enclosed by the geometry, and outside geometry are kept out

- **Issue:** how big of a cube to use
  - A smaller cube leads to finer details
  - A smaller cube also requires more samples



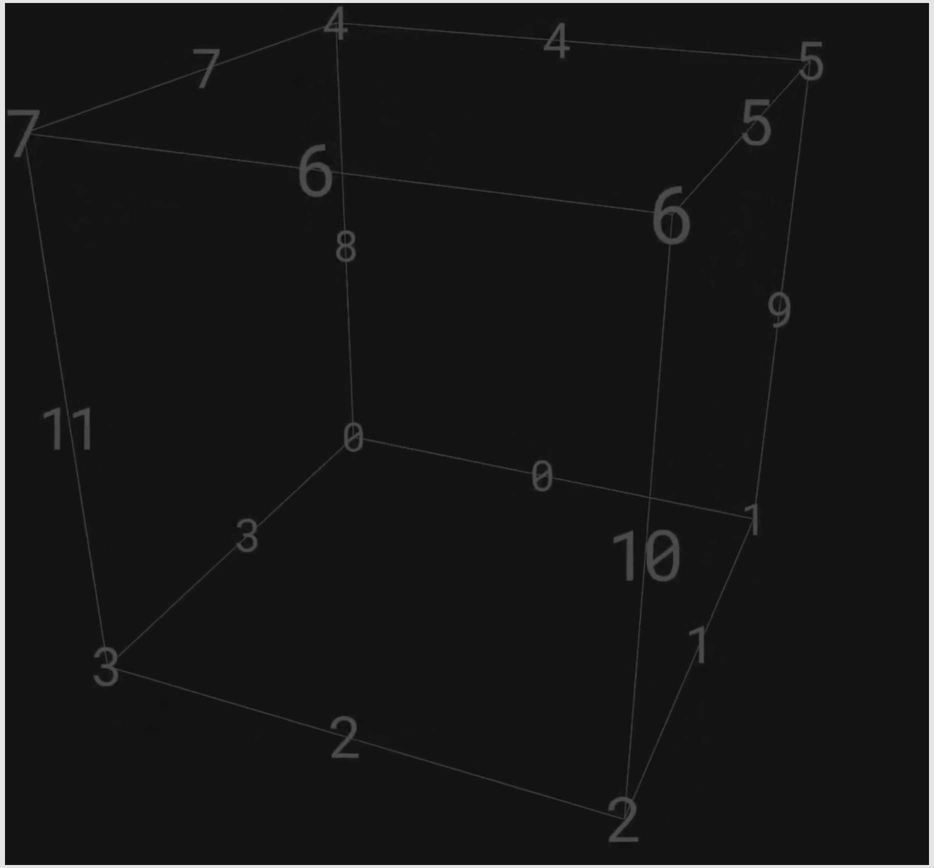
Marching Cubes Vertices

- Each cube has 8 vertices
- Check if each vertex lies inside or outside the implicit function $f(x, y, z)$
  - Can be encoded as an 8-bit number
    - 1 – inside
    - 0 - outside



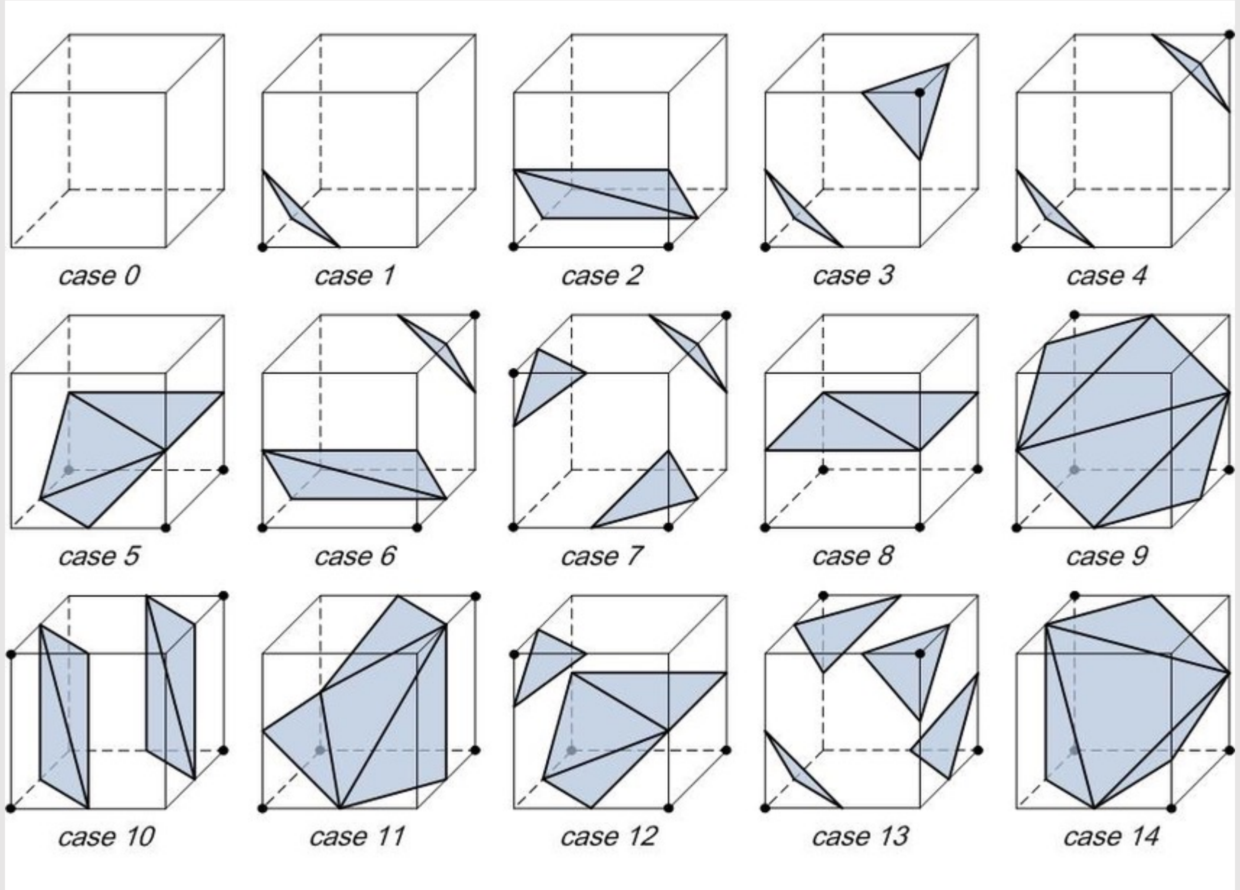
Marching Cubes Edges

- Each cube has 12 edges
- Goal is to create geometry with vertices along the edges of the cube that enclose inside vertices and excludes outside vertices



Marching Cubes Geometry

- $2^8 = 256$ possible configurations
- How do we know which one to use?



Marching Cubes Lookup Table

```
// For each MC case, a list of triangles, specified as triples of edge indices, terminated by -1
const TriangleTable = [
        [ -1 ],
        [ 0, 3, 8, -1 ],
        [ 0, 9, 1, -1 ],
        [ 3, 8, 1, 1, 8, 9, -1 ],
        [ 2, 11, 3, -1 ],
        [ 8, 0, 11, 11, 0, 2, -1 ],
        [ 3, 2, 11, 1, 0, 9, -1 ],
        [ 11, 1, 2, 11, 9, 1, 11, 8, 9, -1 ],
        [ 1, 10, 2, -1 ],
        [ 0, 3, 8, 2, 1, 10, -1 ],
        [ 10, 2, 9, 9, 2, 0, -1 ],
        [ 8, 2, 3, 8, 10, 2, 8, 9, 10, -1 ],
        [ 11, 3, 10, 10, 3, 1, -1 ],
        [ 10, 0, 1, 10, 8, 0, 10, 11, 8, -1 ],
        [ 9, 3, 0, 9, 11, 3, 9, 10, 11, -1 ],
        [ 8, 9, 11, 11, 9, 10, -1 ],
        [ 4, 8, 7, -1 ],
        [ 7, 4, 3, 3, 4, 0, -1 ],
        [ 4, 8, 7, 0, 9, 1, -1 ],
        [ 1, 4, 9, 1, 7, 4, 1, 3, 7, -1 ],
        [ 8, 7, 4, 11, 3, 2, -1 ],
        [ 4, 11, 7, 4, 2, 11, 4, 0, 2, -1 ],
        [ 0, 9, 1, 8, 7, 4, 11, 3, 2, -1 ],
        [ 7, 4, 11, 11, 4, 2, 2, 4, 9, 2, 9, 1, -1 ],
        [ 4, 8, 7, 2, 1, 10, -1 ],
        [ 7, 4, 3, 3, 4, 0, 10, 2, 1, -1 ],
        [ 10, 2, 9, 9, 2, 0, 7, 4, 8, -1 ],
        [ 10, 2, 3, 10, 3, 4, 3, 7, 4, 9, 10, 4, -1 ],
        [ 1, 10, 3, 3, 10, 11, 4, 8, 7, -1 ],
        [ 10, 11, 1, 11, 7, 4, 1, 11, 4, 1, 4, 0, -1 ],
        [ 7, 4, 8, 9, 3, 0, 9, 11, 3, 9, 10, 11, -1 ],
        [ 7, 4, 11, 4, 9, 11, 9, 10, 11, -1 ],
        [ 9, 4, 5, -1 ],
        [ 9, 4, 5, 8, 0, 3, -1 ],
        [ 4, 5, 0, 0, 5, 1, -1 ],
        [ 5, 8, 4, 5, 3, 8, 5, 1, 3, -1 ],
```

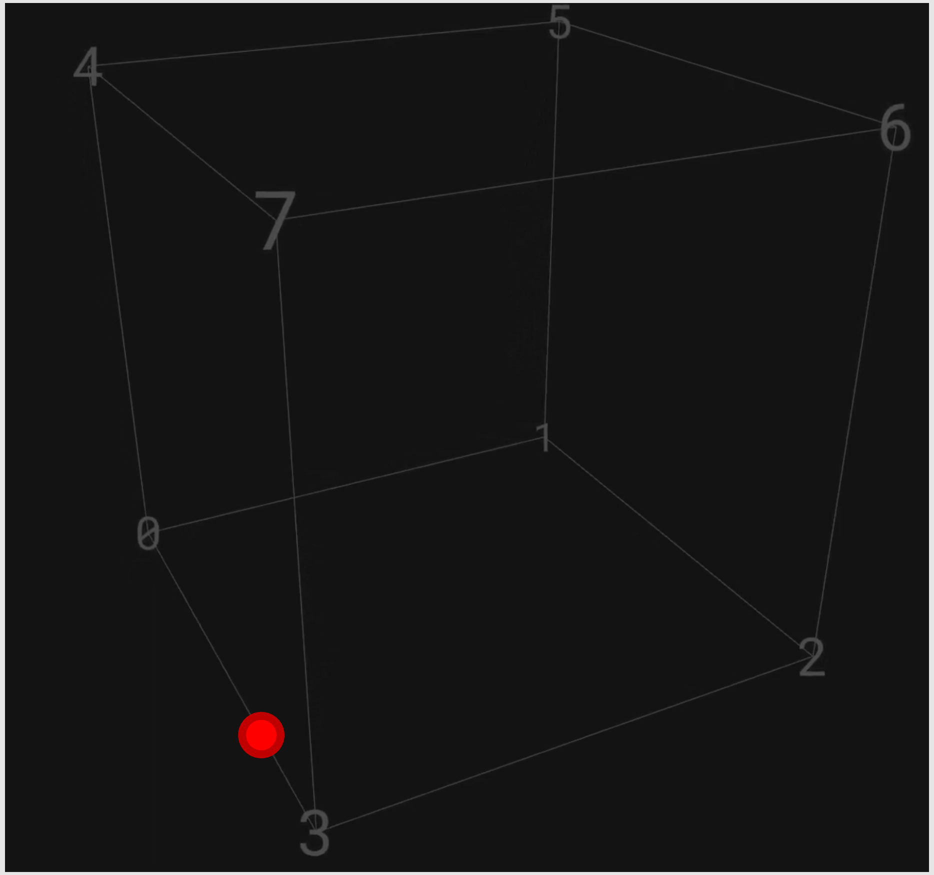
- Just use a lookup table!

Marching Cubes Linear Interpolation

- **Issue:** lookup table only tells us on what edges to place vertices and how to connect them
  - Does not tell us the specific location of vertices

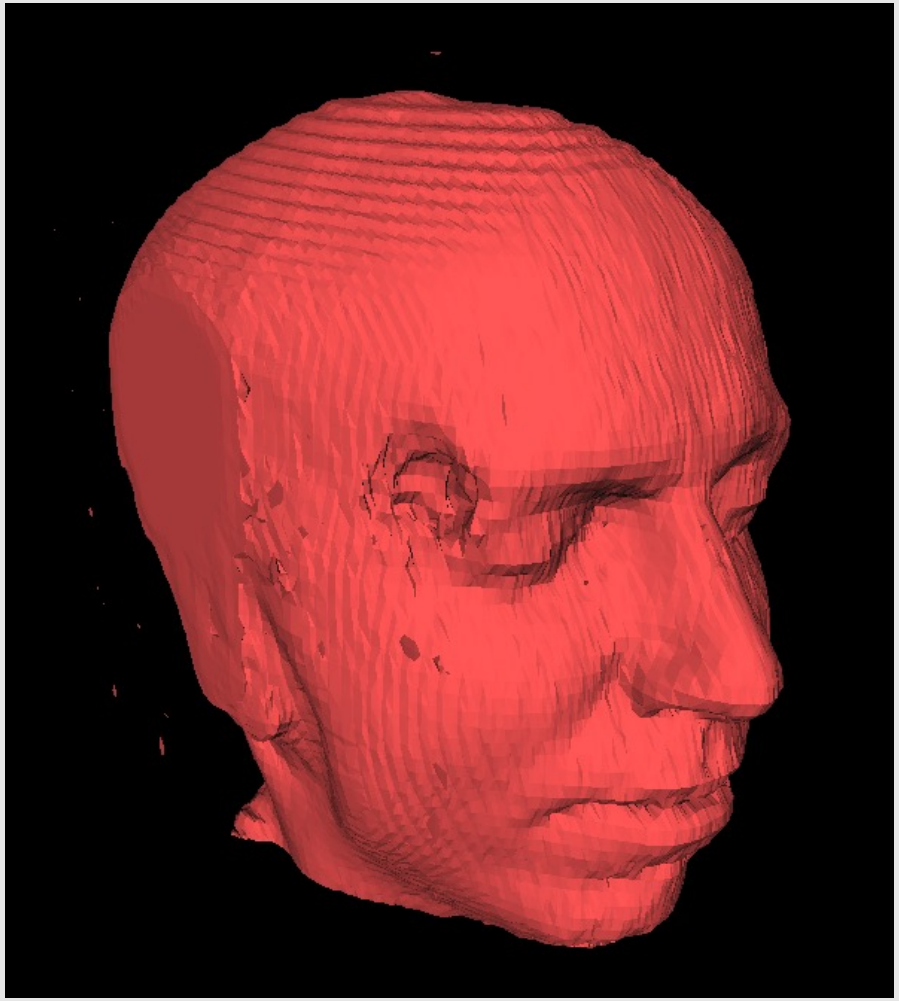
- When placing vertices, can linearly interpolate them on the edges depending on the evaluated values on the cube vertices

- Example:
  - $f(x_0, y_0, z_0) = -0.75$
  - $f(x_3, y_3, z_3) = +0.25$
    - Vertex is placed ¼ distance away from corner 3, ¾ distance from corner 0



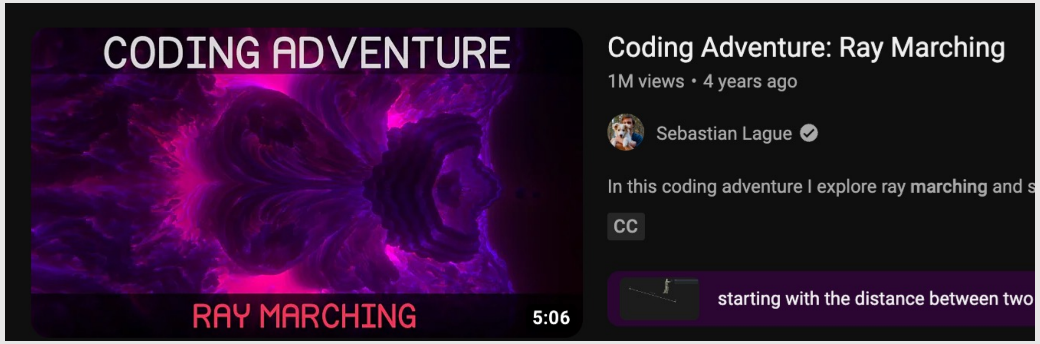
Marching Cubes Examples

- **Issue:** very cube-like
  - Easy to see cube artifacts
- How to fix?
  - Run refinement
  - Run denoising
  - Run remeshing



Marching Cubes Application

- Terrain generation
- Implicitly generate terrain with algebraic surfaces and noise
  - Convert to explicit mesh for easy rendering



- ~~Marching Cubes~~

- Signed Distance Fields

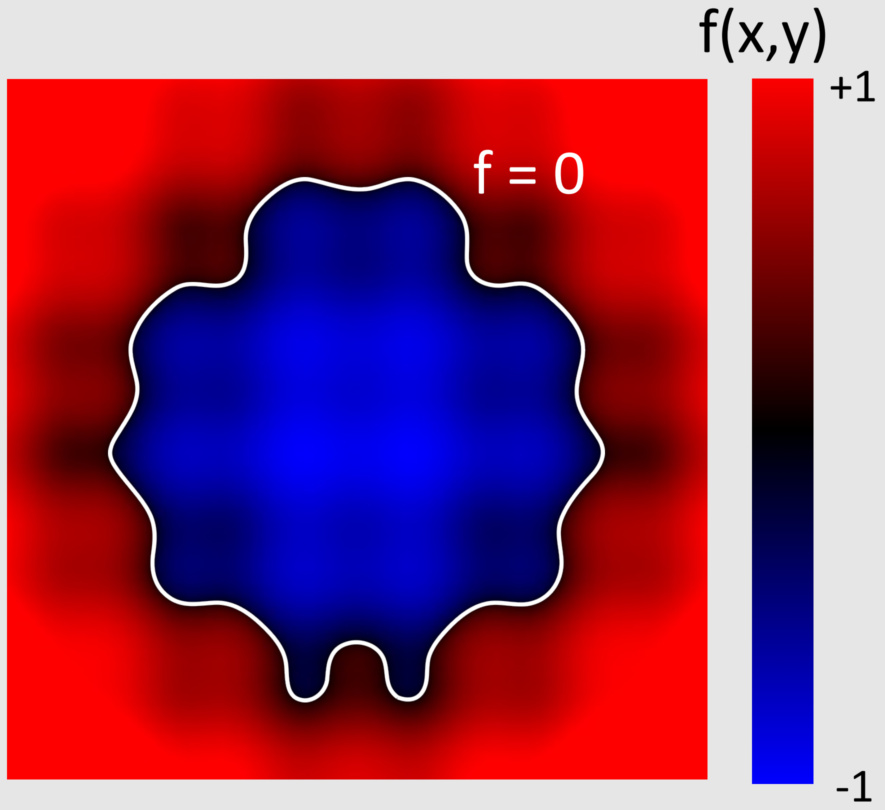
- NERFs

How do we convert explicit geometry to implicit geometry?

Signed Distance Fields

- Signed distance fields are implicit functions $f(x, y, z)$ that tell us the sign (inside/outside) and the distance away from the boundary
  - Gradient $\nabla f(x, y, z)$ makes finding the boundary easier

- SDFs make it easy to check where and how far a point is from a surface

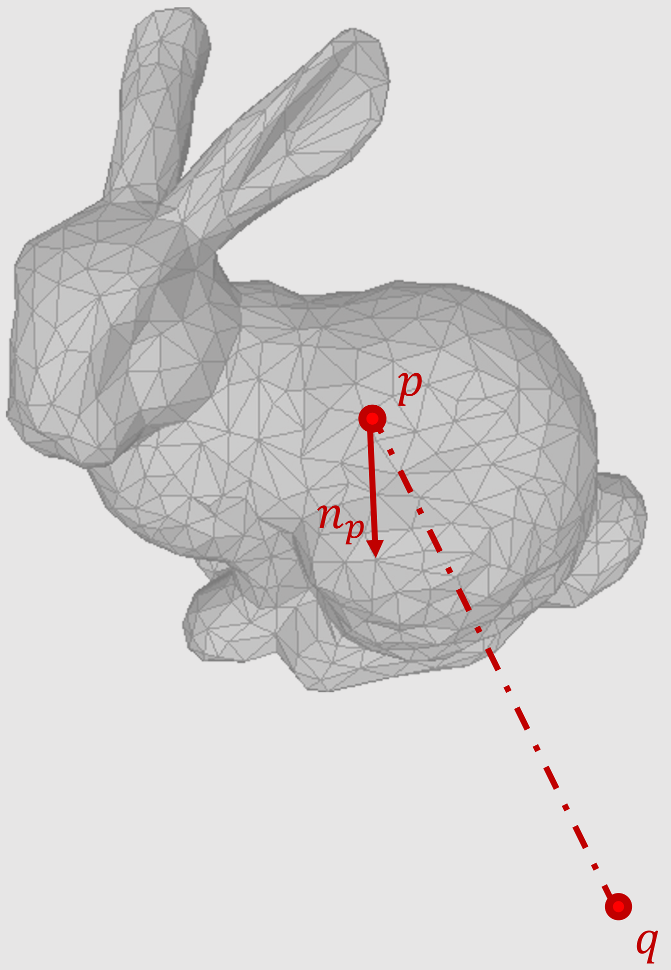


Converting Mesh To SDF

- **Idea:** SDF of a mesh should be proportional to the closest point p on a mesh to some query point q
- **Issue:** how to accelerate computing the closest point on the mesh
  - Accelerated geometric queries

```
For a given query point q:
    Compute the closest point on the mesh p
    Compute the normal np for p
    Project the vector (q-p) onto np
```

Converting Mesh To SDF

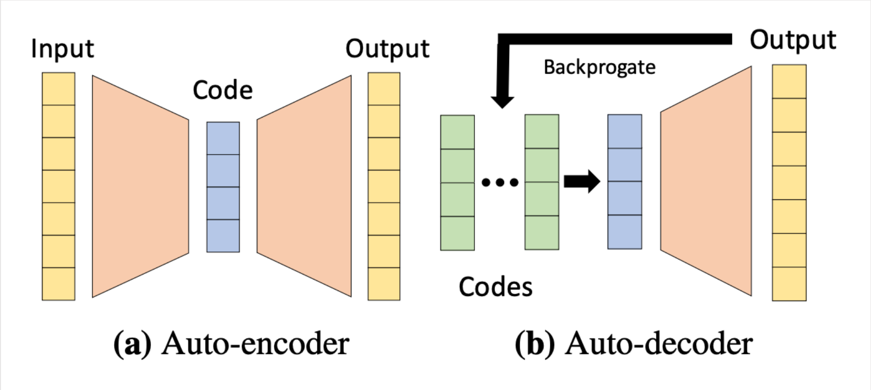


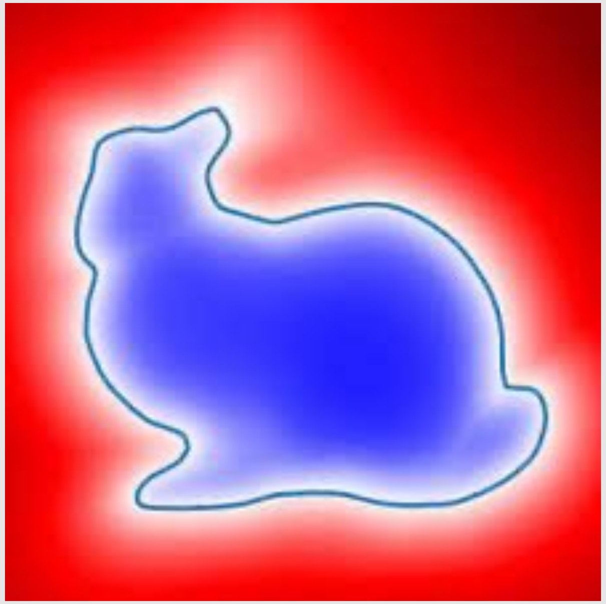

```
For a given query point q:

    Compute the closest point on the mesh p

    Compute the normal np for p

    Project the vector (q-p) onto np
```

- **Distance** encoded by $|q - p|$
- **Sign** encoded by $(q - p) \cdot n_p$

Neural SDFs

- Constructing a SDF can be difficult/expensive
- Throw a bunch of evaluated samples into an autoencoder
  - Learn a SDF representation of the data
- Neural net maps (x,y,z) to a signed distance
  - Can be used same way as an SDF

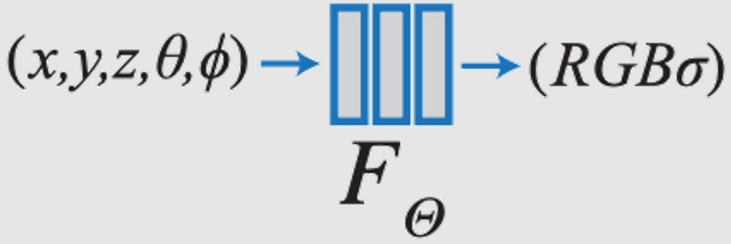


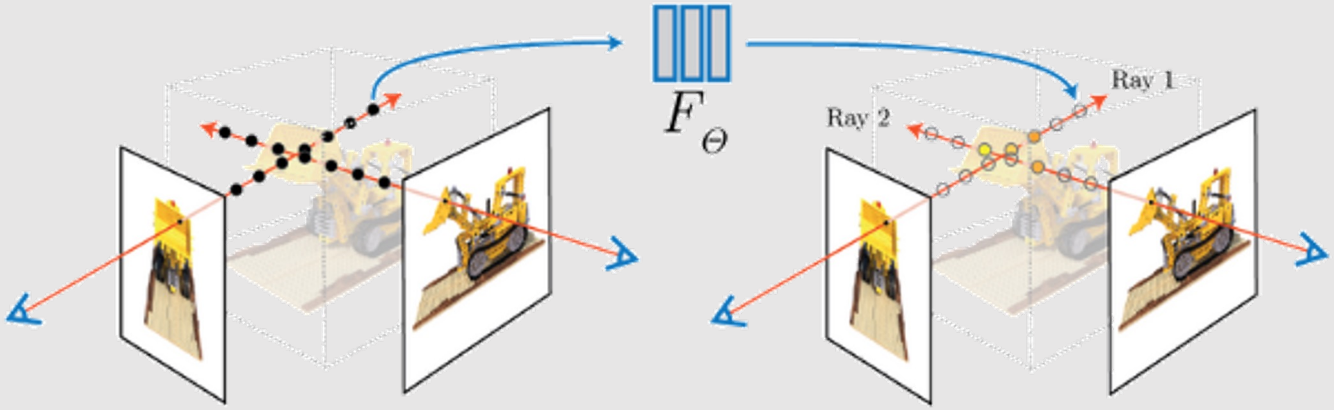


DeepSDF (2019) Park et al.

- ~~Marching Cubes~~
- ~~Signed Distance Fields~~
- NERFs

Neural Radiance Fields





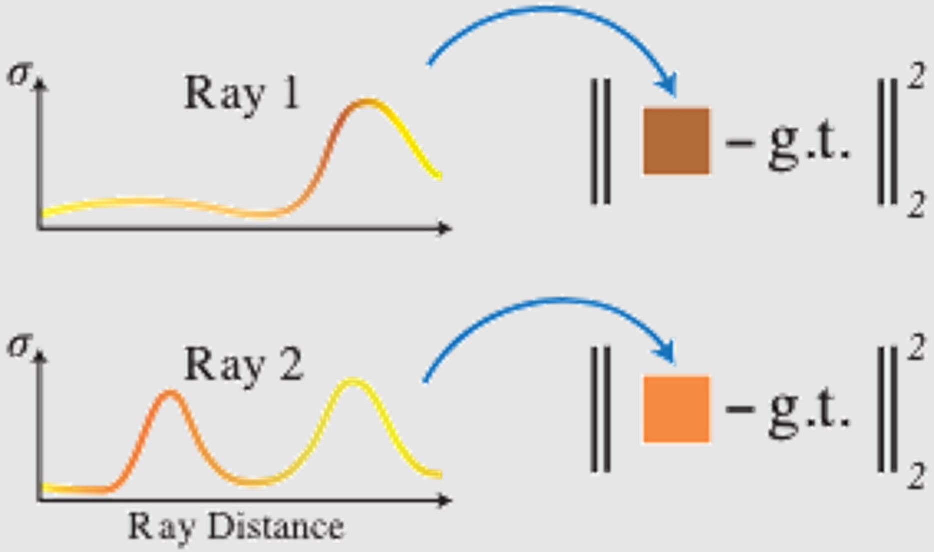
B. Mildenhall (2020)

- Train neural network F on multiple images
  - Training data: (x,y,z) of pixel + view angle + RGB
    - Depth of pixel must be known
  - Test data: (x,y,z) of requested pixel + view angle
    - Outputs RGB
    - What if we don't know the z we want?
      - Just grab the nearest z
      - No different than 0-depth ray tracing

    - To train properly, need multiple images each from a different view angle
      - **Key Assumption:** images must rotate about a fixed origin
        - Hence only 2 d.o.f with view angle

Neural Radiance Fields

- We are building a field of radiance values
  - In case that wasn't clear :)
- Output RGB depends on depth
  - Think of it as a slice in a ray's direction
  - Optimize known (depth, RGB) pairs along a ray
    - Model learns to interpolate to unseen RGB values
- **Key Assumption:** lighting should remain constant between scenes
  - **Q: What would the distribution look like if not?**



B. Mildenhall (2020)

View-Dependent Appearances

- Some changes in lighting are inevitable
  - Mirror reflections
  - Glass refractions
  - Anisotropic materials
- **Idea:** treat these as normal pixels
  - View-dependent lighting will get baked into the model
    - Recall, when evaluating our model, we pass in the view direction
      - View-dependent lighting will only be present if same view angle passed into neural network

B. Mildenhall (2020)

Extra Features

B. Mildenhall (2020)

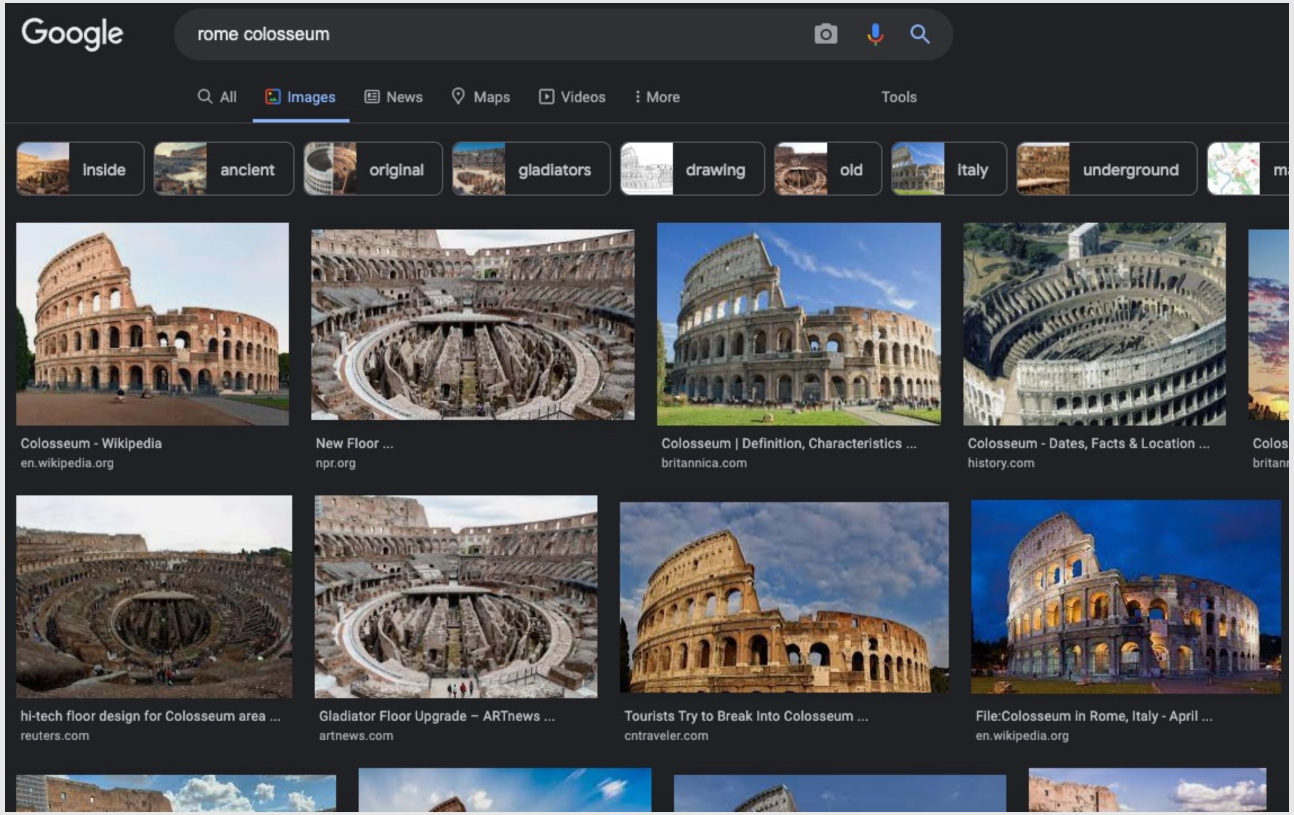
- What can we use NERFs for?
  - Depth Maps
  - Image Relighting
  - Object insertion

https://www.matthewtancik.com/nerf

NeRF In The Wild

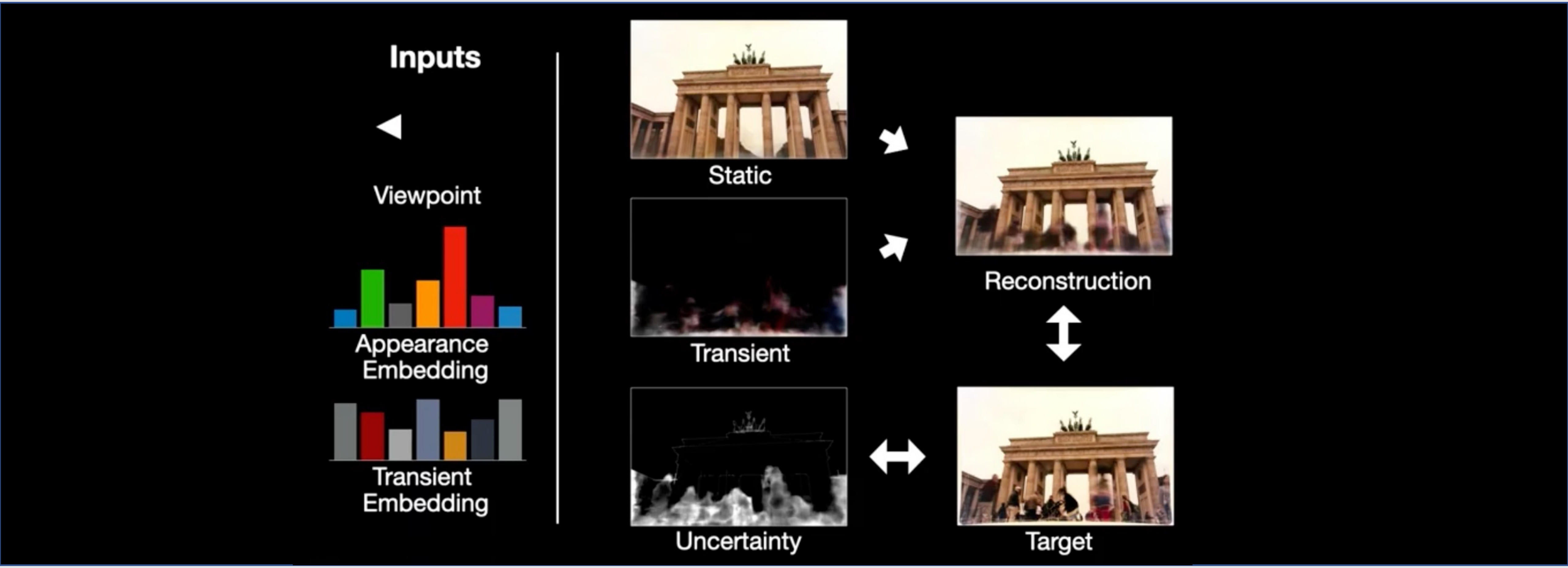
R. Martin-Brualla (2021) Google

Image Databases



- **Key Idea:** photos for scene reconstruction are not always taken by the same user in the same conditions
  - Dealing with large database of image requires understanding of static properties and transient properties
    - Transient properties include:
      - People and object occlusions
      - Weather
      - Time of day

Removing Transient Features



- Learn Appearance and transient embedding that minimizes reconstruction error

Removing Transient Features

- Modify appearance embedding to change weather and lighting
  - Allows multiple photos to share same global properties
- Now we have an image set to perform NeRF on
  - Recall viewport given