UC San Diego

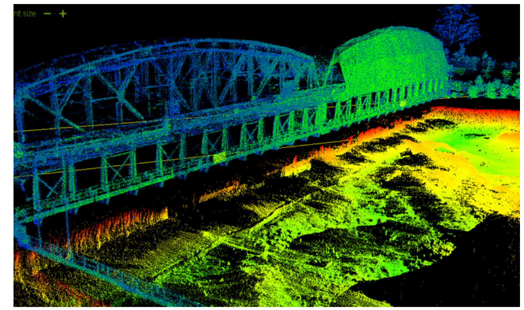
Machine Learning meets Geometry

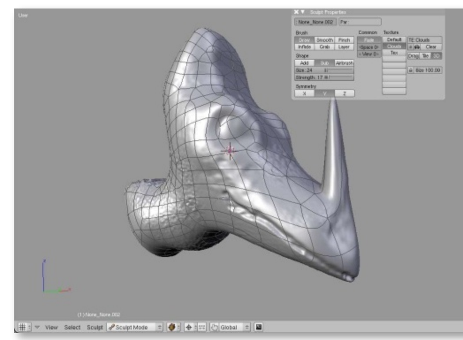
L4: Mesh and Point Cloud

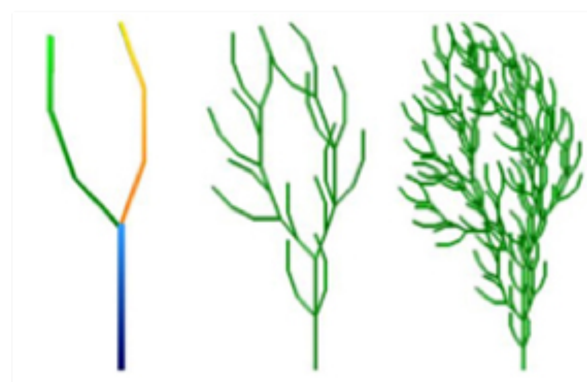
Hao Su

Shape Representation: Origin- and Application-Dependent

- Acquired real-world objects
- Modeling “by hand”
- Procedural modeling
- …

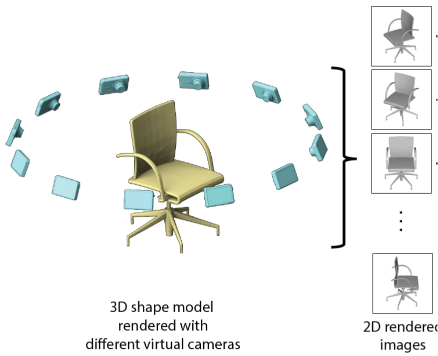






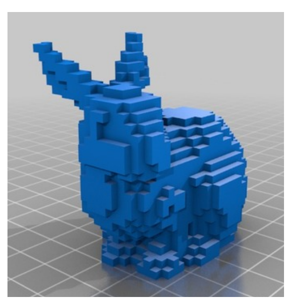
Other than parametric representations, we also study these in this course:

Rasterized form (regular grids)



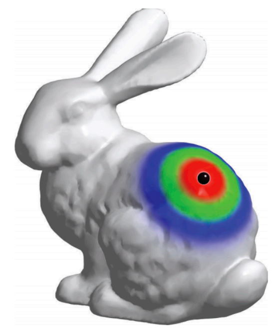
Multi-view

Depth Map

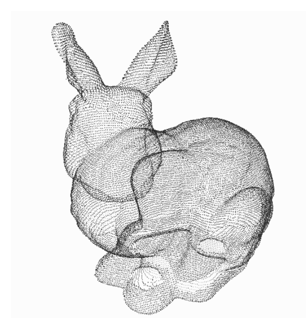


Volumetric

Geometric form (irregular)



Mesh

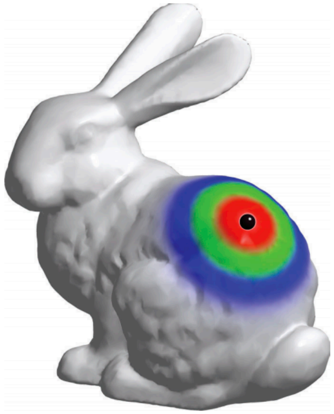


Point Cloud

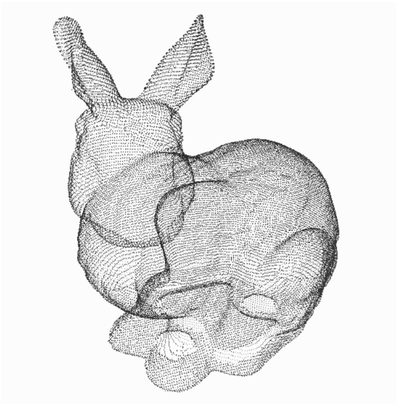
$$F(x) = 0$$

Implicit Shape

Agenda



Mesh



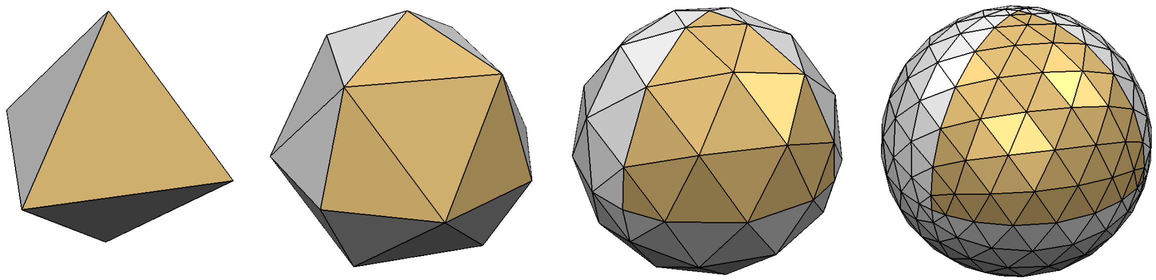
Point Cloud

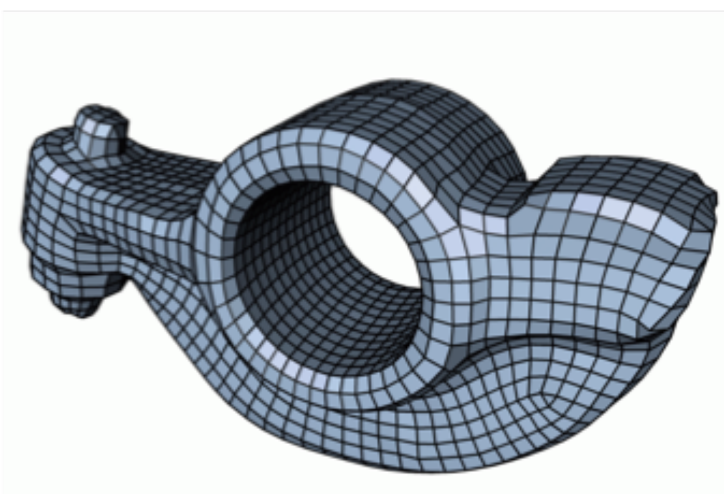
Polygonal Meshes

- Representation
- Storage
- Curvature Computation

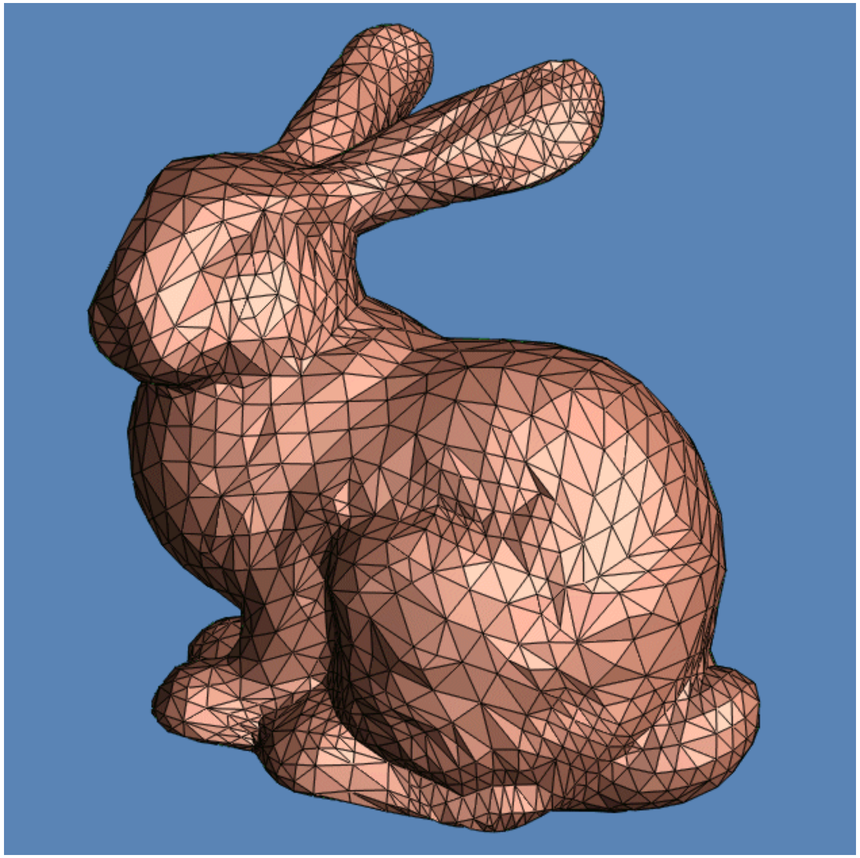
Polygonal Meshes

- Piece-wise Linear Surface Representation





Triangle Mesh



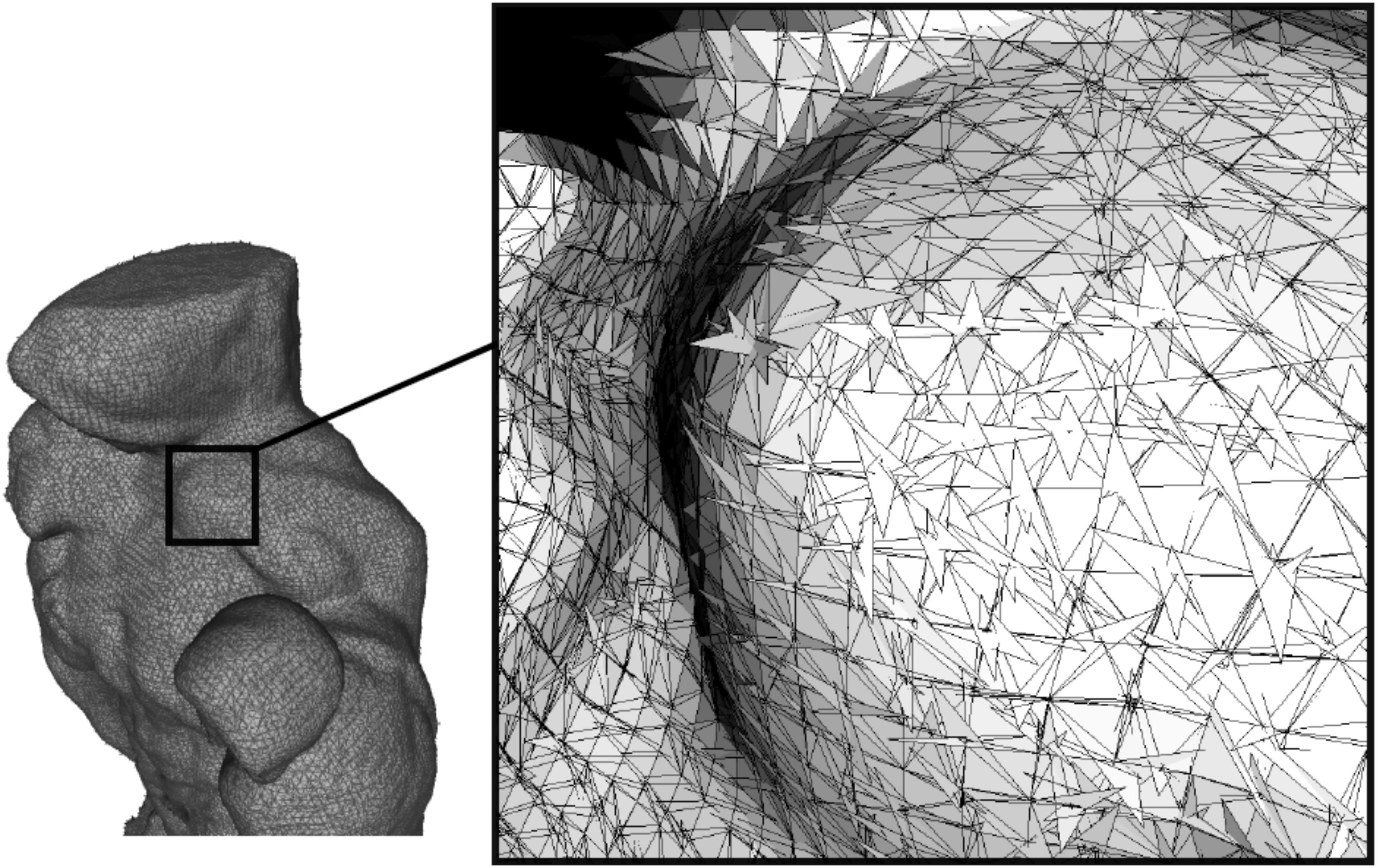
http://graphics.stanford.edu/data/3Dscanrep/stanford-bunny-cebal-ssh.jpg
http://www.stat.washington.edu/wxs/images/BUNMID.gif

Triangle Mesh

$$V = \{v_1, v_2, \ldots, v_n\} \subset \mathbb{R}^3$$
$$E = \{e_1, e_2, \ldots, e_k\} \subseteq V \times V$$
$$F = \{f_1, f_2, \ldots, f_m\} \subseteq V \times V \times V$$

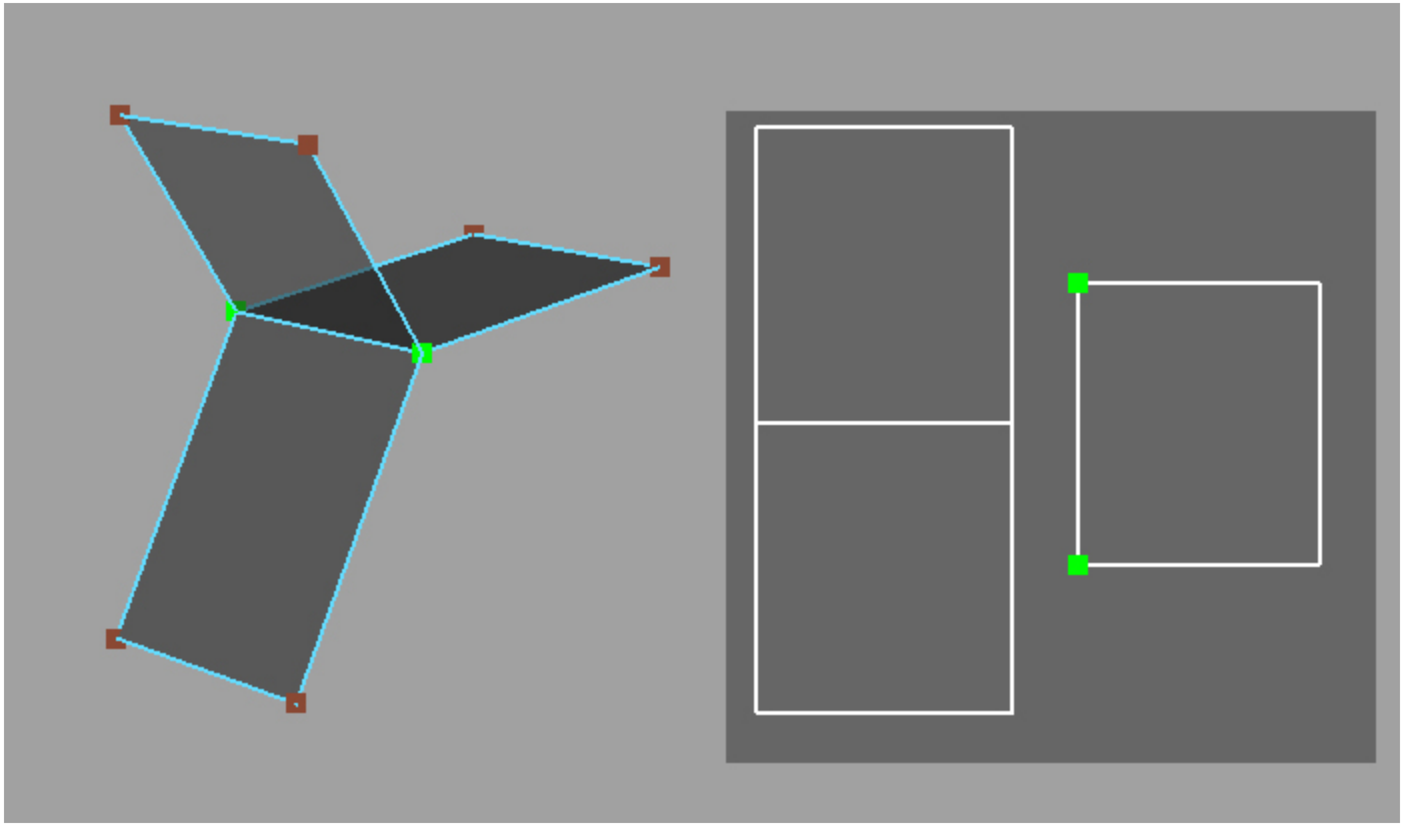
Plus manifold conditions

Bad Surfaces



http://igl.ethz.ch/projects/parameterization/rangemap-param/rangemap-param.pdf

Nonmanifold Edge

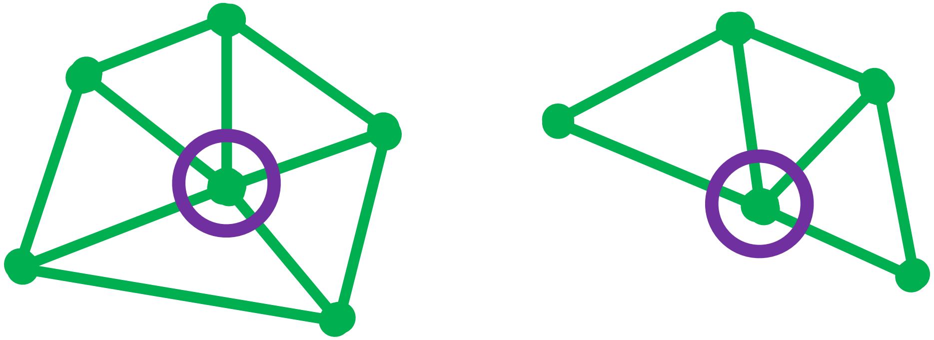


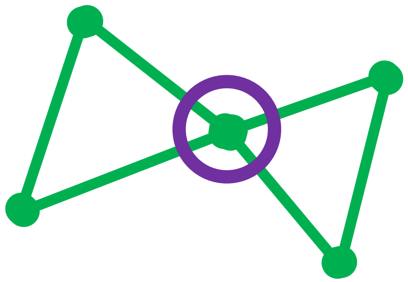
http://blog.mixamo.com/wp-content/uploads/2011/01/nonmanifold.jpg

Manifold Mesh

1. Each edge is incident to one or two faces

2. Faces incident to a vertex form a closed or open fan

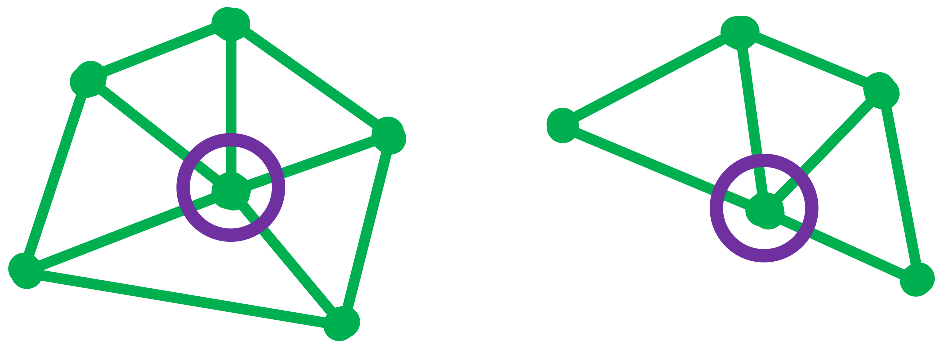


This is not a fan: 

http://www.cs.mtu.edu/~shene/COURSES/cs3621/SLIDES/Mesh.pdf

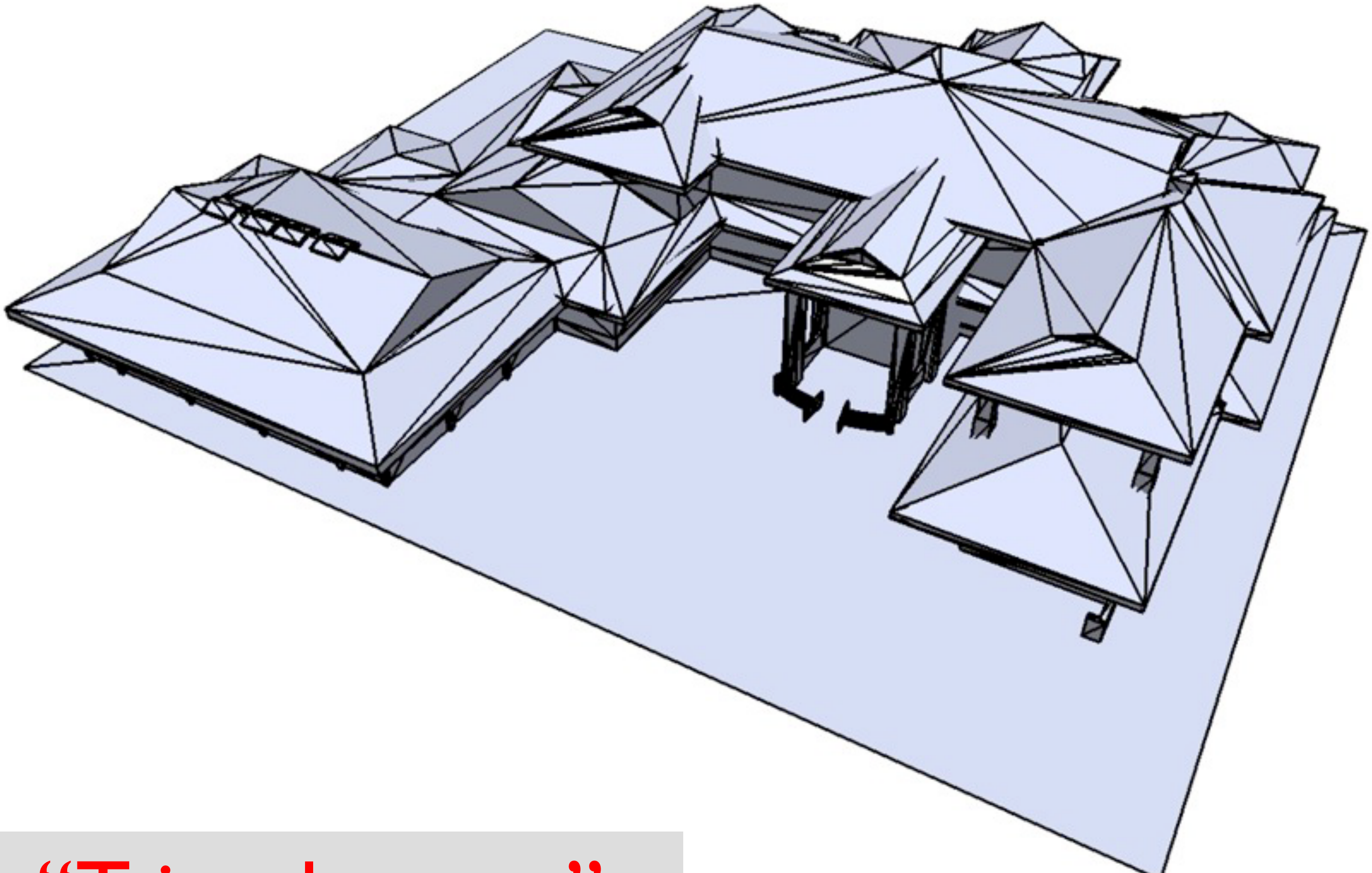
Manifold Mesh

1. Each edge is incident to one or two faces
2. Faces incident to a vertex form a closed or open fan



Assume meshes are manifold
(for now)

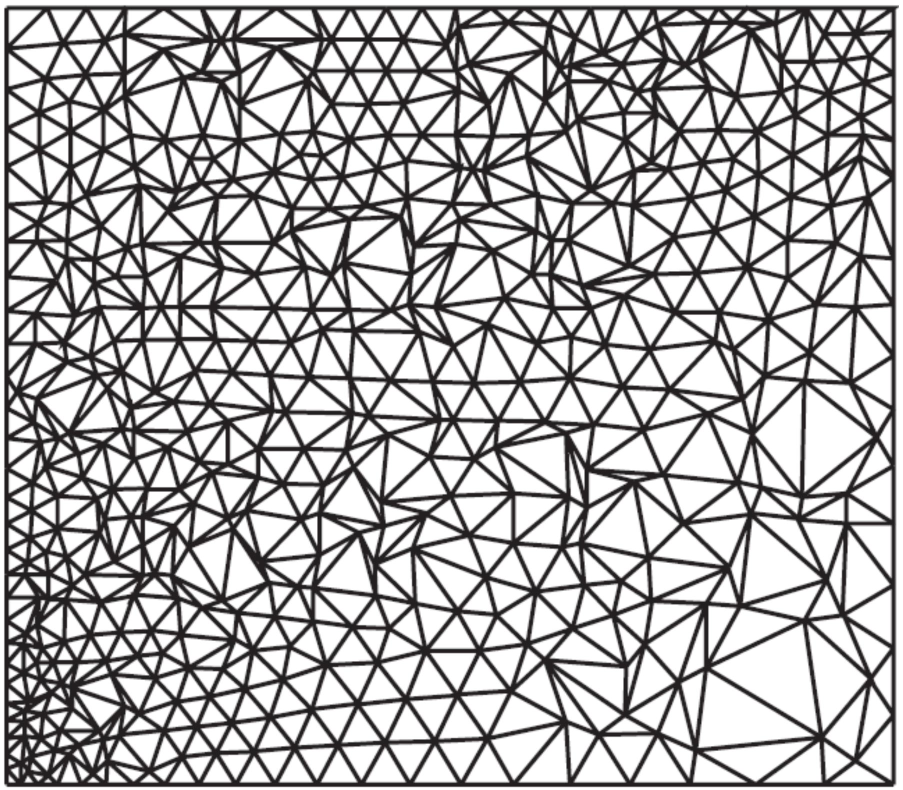
http://www.cs.mtu.edu/~shene/COURSES/cs3621/SLIDES/Mesh.pdf

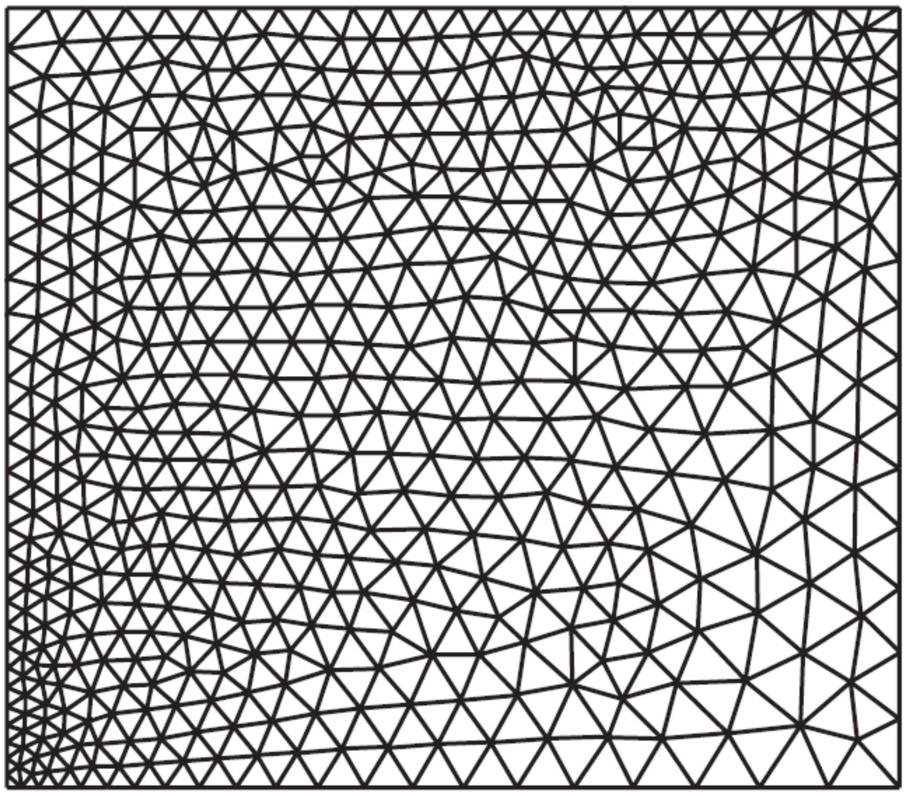


“Triangle soup”

http://www.pmp-book.org/download/slides/Representations.pdf

Bad Meshes

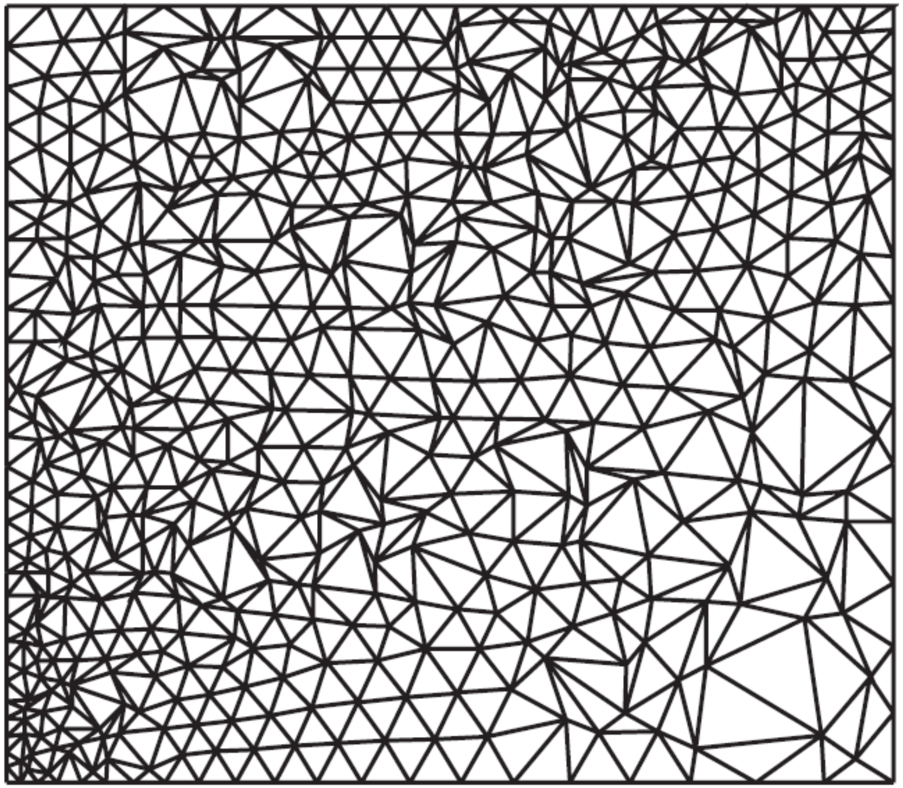




Nonuniform
areas and angles

http://www.sciencedirect.com/science/article/pii/S0168874X06000795

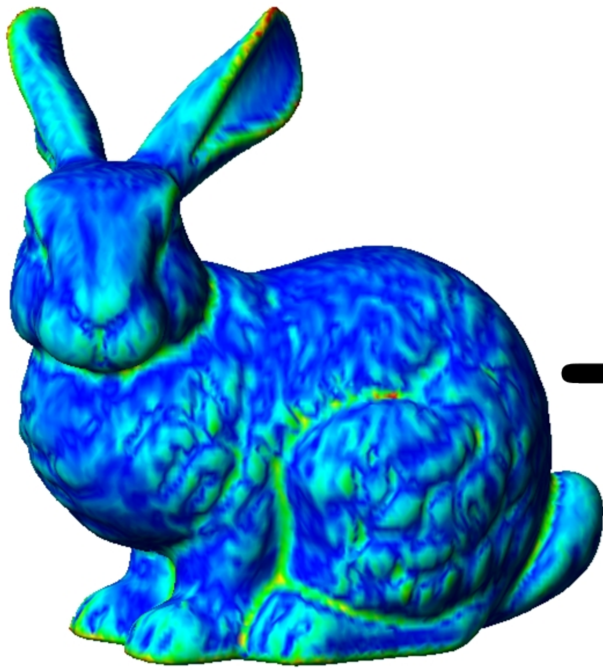
Why is Meshing an Issue?



How do you interpret
one value per vertex?

http://www.sciencedirect.com/science/article/pii/S0168874X06000795

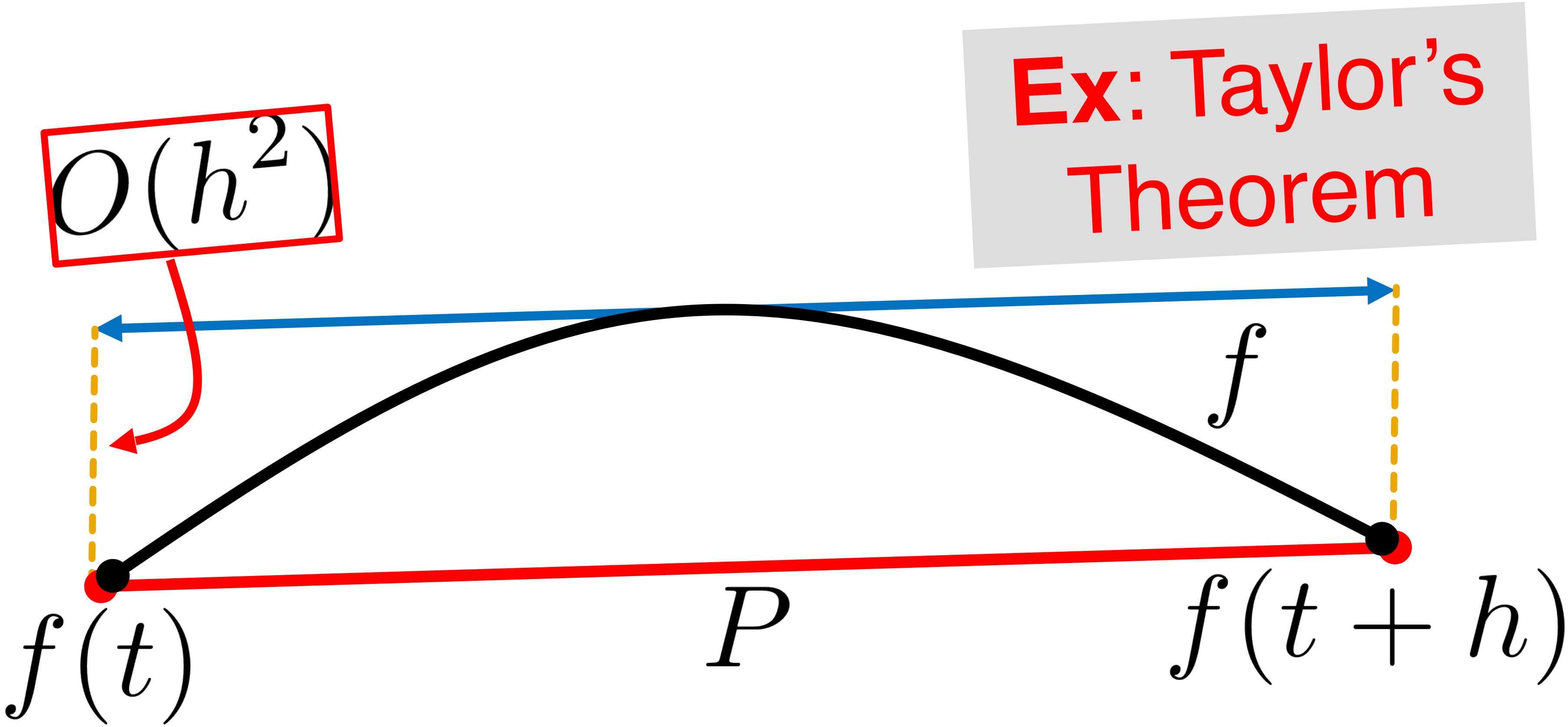
Assume Storing Scalar Functions on Surface

$f :$  $\longrightarrow \mathbb{R}$

http://www.ieeta.pt/polymeco/Screenshots/PolyMeCo_OneView.jpg

Map points to real numbers

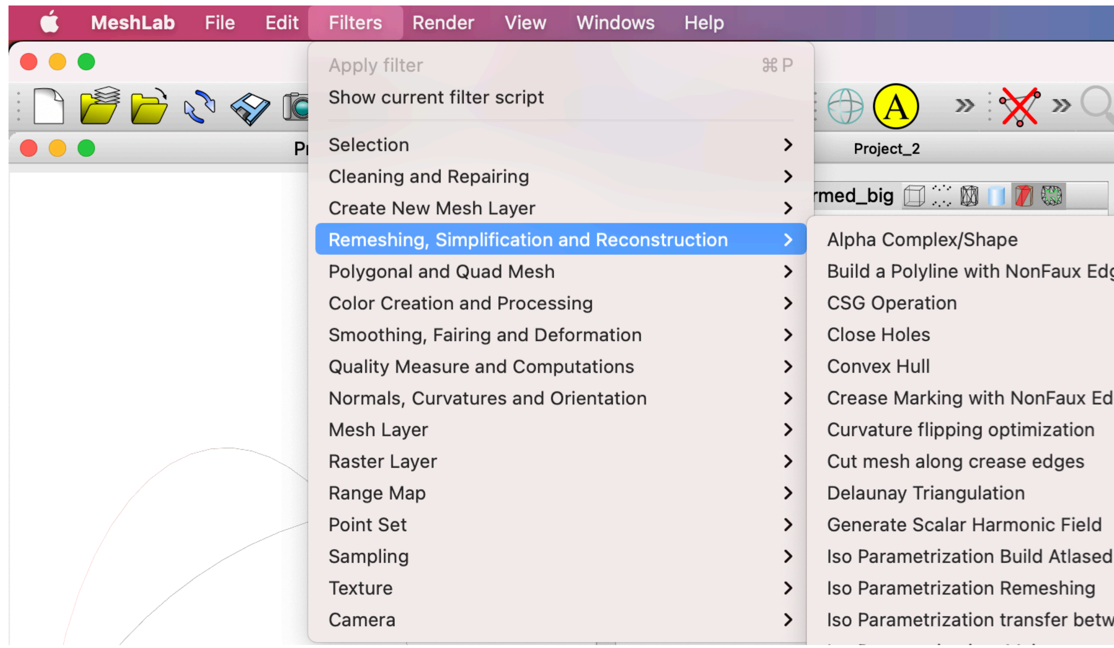
Approximation Properties



f : functions defined at vertices
(e.g., Gaussian curvature)

Techniques to Improve Mesh Quality

- Cleaning
- Repairing
- Remeshing
- …



Polygonal Meshes

- Representation
- Storage
- Curvature Computation

Data Structures for Surfaces

- What should be stored?
  - Geometry: 3D coordinates
  - Topology
  - Attributes
    - Normal, color, texture coordinates
    - Per vertex, face, edge

Simple Data Structures: Triangle List

- STL format (used in CAD)
- Storage
  - Face: 3 positions
- No connectivity information

Triangles			
0	x0	y0	z0
1	x1	x1	z1
2	x2	y2	z2
3	x3	y3	z3
4	x4	y4	z4
5	x5	y5	z5
6	x6	y6	z6
...	...	...	...

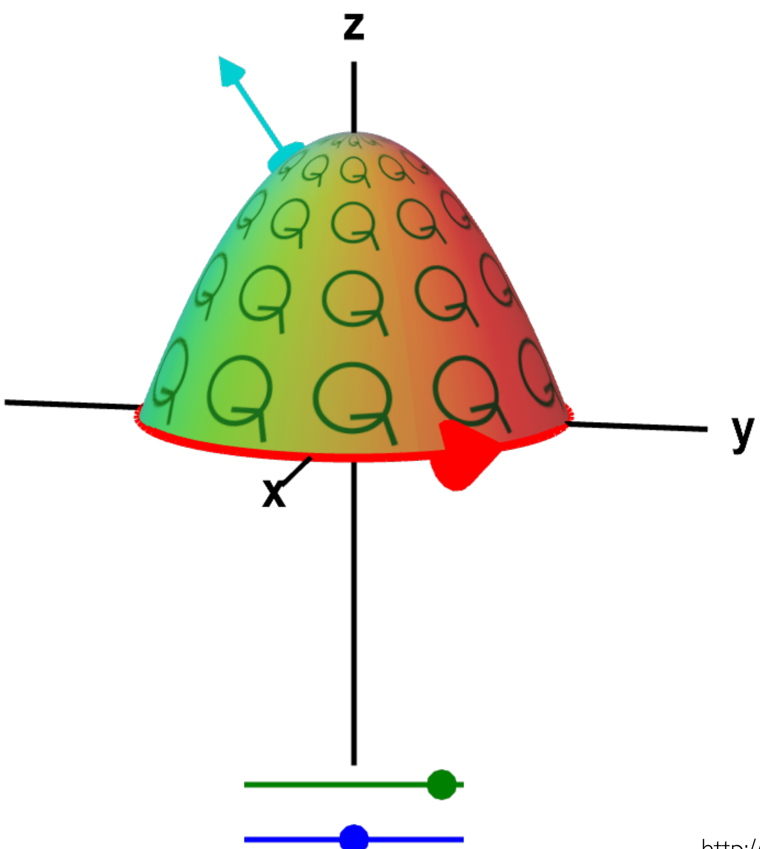
Simple Data Structures: Indexed Face Set

- Used in formats
  - OBJ, OFF, WRL
- Storage
  - Vertex: position
  - Face: vertex indices
  - Convention is to save vertices in counter-clockwise order for normal computation

Vertices			
v0	x0	y0	z0
v1	x1	x1	z1
v2	x2	y2	z2
v3	x3	y3	z3
v4	x4	y4	z4
v5	x5	y5	z5
v6	x6	y6	z6
...	...	...	...

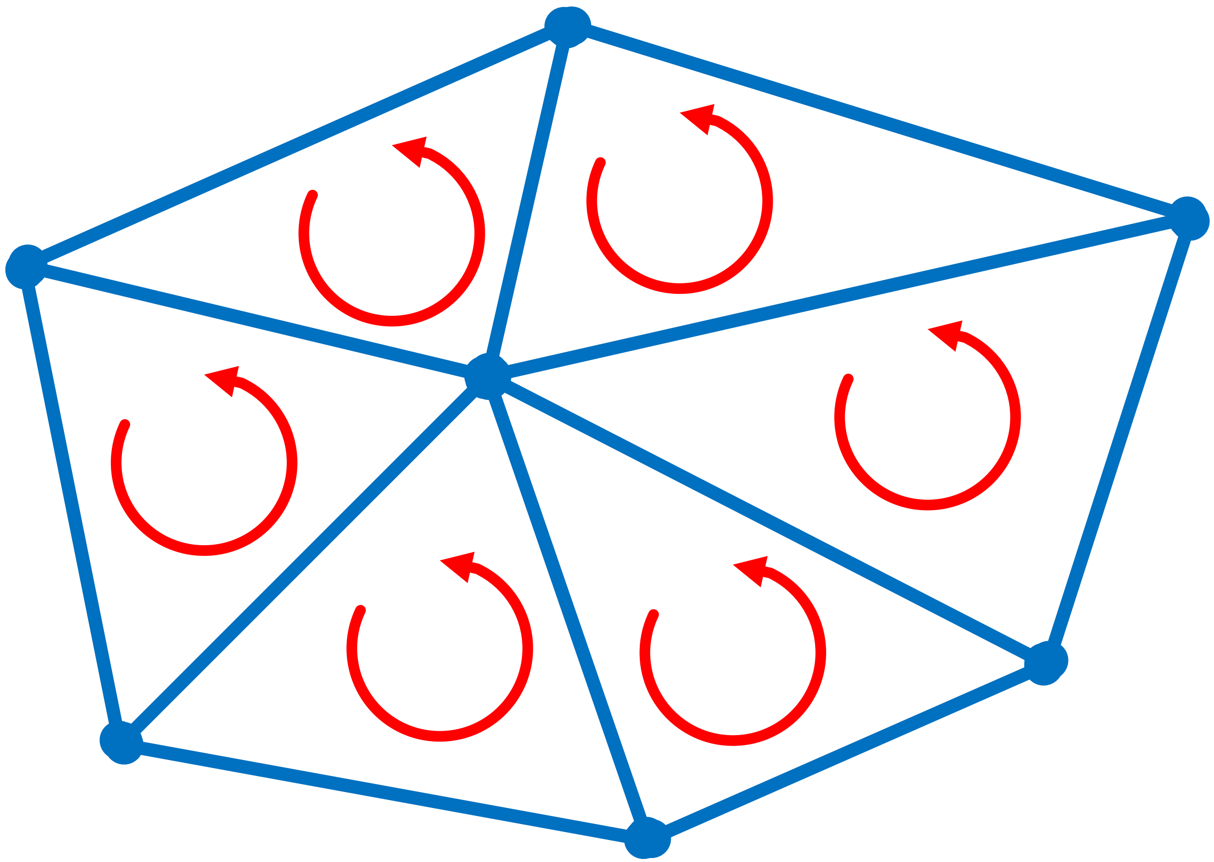
Triangles			
t0	v0	v1	v2
t1	v0	v1	v3
t2	v2	v4	v3
t3	v5	v2	v6
...	...	...	...

Right-Hand Rule

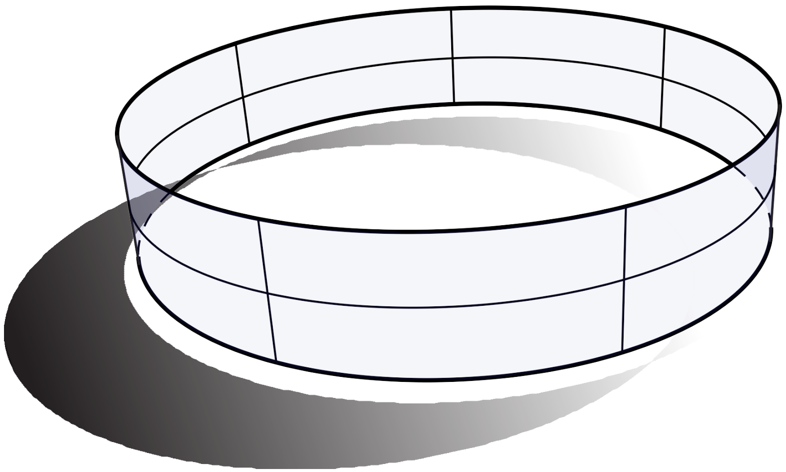


http://viz.aset.psu.edu/gho/sem_notes/3d_fundamentals/html/3d_coordinates.html
http://mathinsight.org/stokes_theorem_orientation

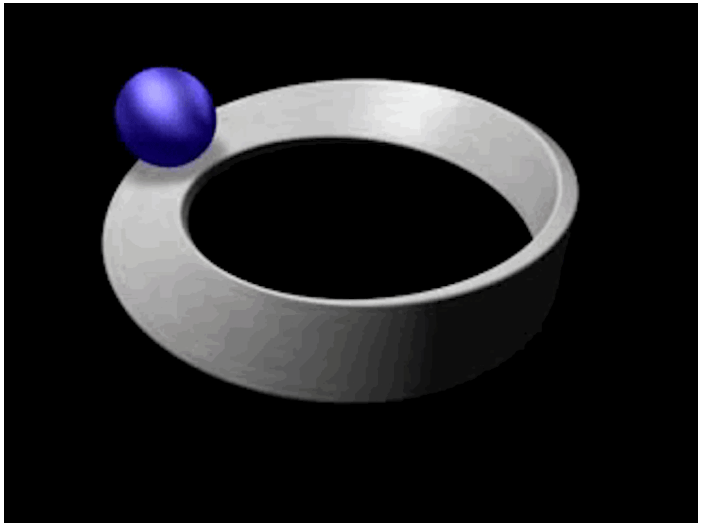
Normal Computation



Orientability



orientable



non-orientable

http://blog.lightformshop.com/the-kepler-collection-nemo-math-inspired-wonder/mobius-strip-sphere-2/

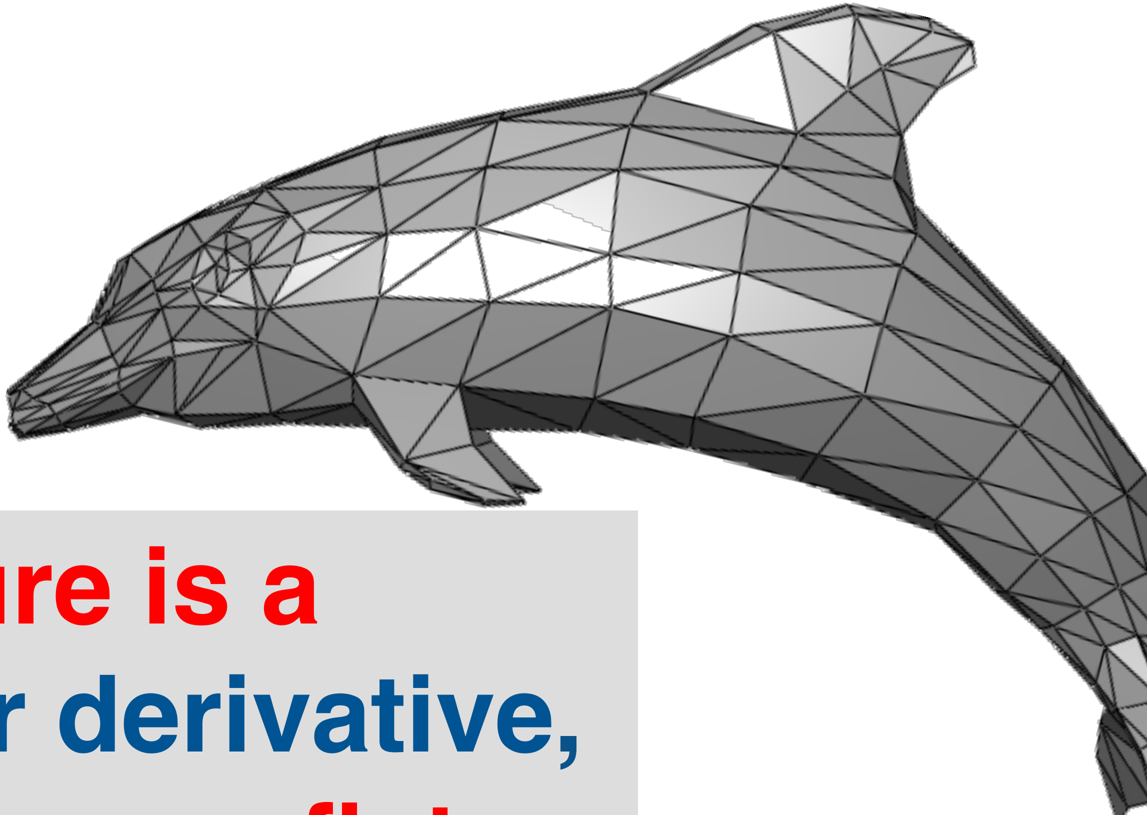
Summary of Polygonal Meshes

- Polygonal meshes are piece-wise linear approximation of smooth surfaces
- Good triangulation is important (manifold, equi-length)
- Vertices, edges, and faces are basic elements
- While real-data 3D are often point clouds, meshes are quite often used to visualize 3D and generate ground truth for machine learning algorithms

Polygonal Meshes

- Representation
- Storage
- Curvature Computation

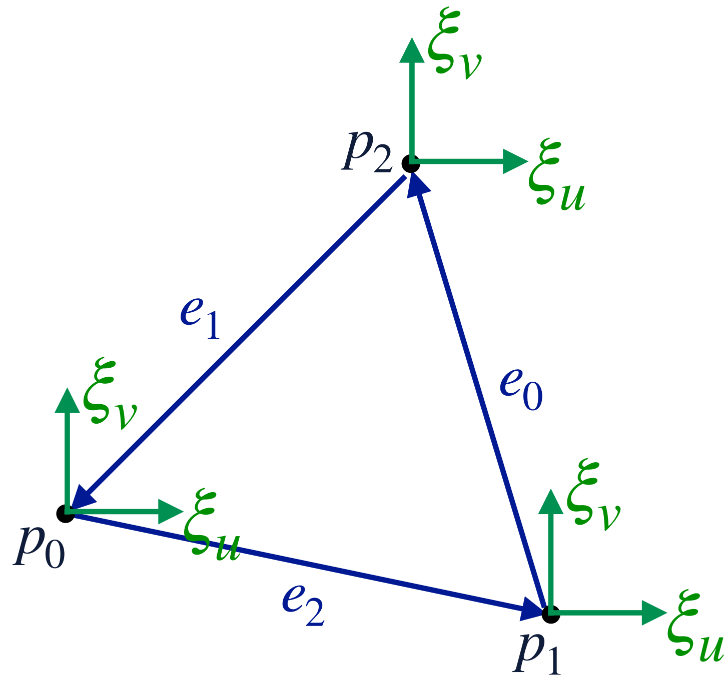
Challenge on Meshes



Curvature is a second-order derivative, but triangles are flat.

http://upload.wikimedia.org/wikipedia/commons/f/fb/Dolphin_triangle_mesh.png

Rusinkiewicz's Method



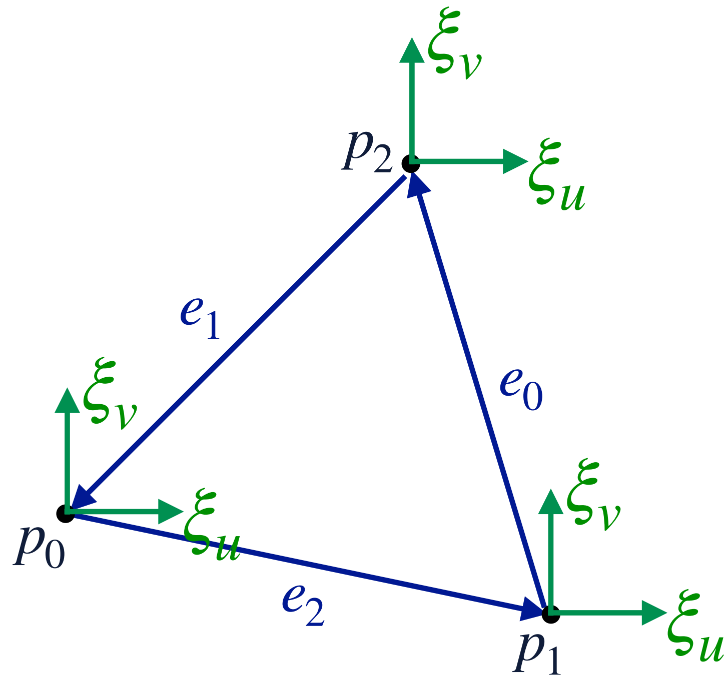
Assume a local $f : U \to \mathbb{R}^3$ at a small triangle

Assume that $\mathbf{T}_{p_i}$'s are roughly parallel

Assume that $Df \begin{bmatrix} u \\ v \end{bmatrix} = u \overrightarrow{\xi}_u + v \overrightarrow{\xi}_v$, i.e., $Df = \left[\overrightarrow{\xi}_u, \overrightarrow{\xi}_v \right]$

(We pick a pair of orthonormal vectors in $\mathbf{T}_{p_i}$ to build a local frame)

Rusinkiewicz's Method



Recall shape operator: $DN = Df \cdot S$.

$$\because Df = \left[\overrightarrow{\xi}_u, \overrightarrow{\xi}_v\right], \therefore S = Df^T DN$$

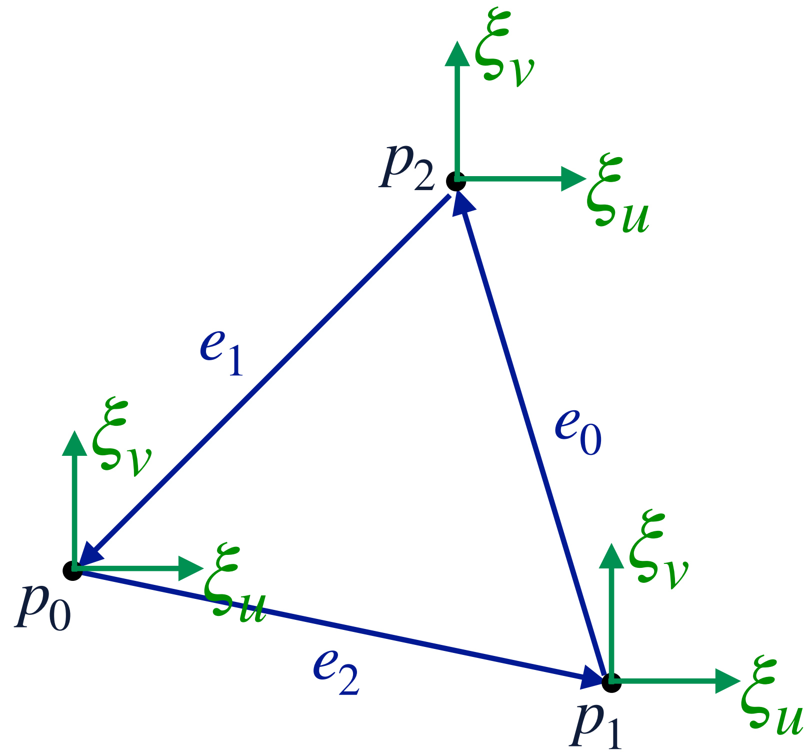
If we have S, we can compute principal curvatures!

How to estimate S?

$$Df^T \left(DN \begin{bmatrix} u \\ v \end{bmatrix} \right) \approx Df^T \Delta \overrightarrow{n} \implies S \begin{bmatrix} u \\ v \end{bmatrix} \approx Df^T \Delta \overrightarrow{n}$$

$\because \; Df \begin{bmatrix} u \\ v \end{bmatrix} = Y \in \mathbf{T}(\mathbb{R}^3)$ and $Df = [\overrightarrow{\xi}_u, \overrightarrow{\xi}_v] \quad \therefore \begin{bmatrix} u \\ v \end{bmatrix} = Df^T Y$

$$\therefore \; S[Df]^T Y \approx [Df]^T \Delta \overrightarrow{n}$$



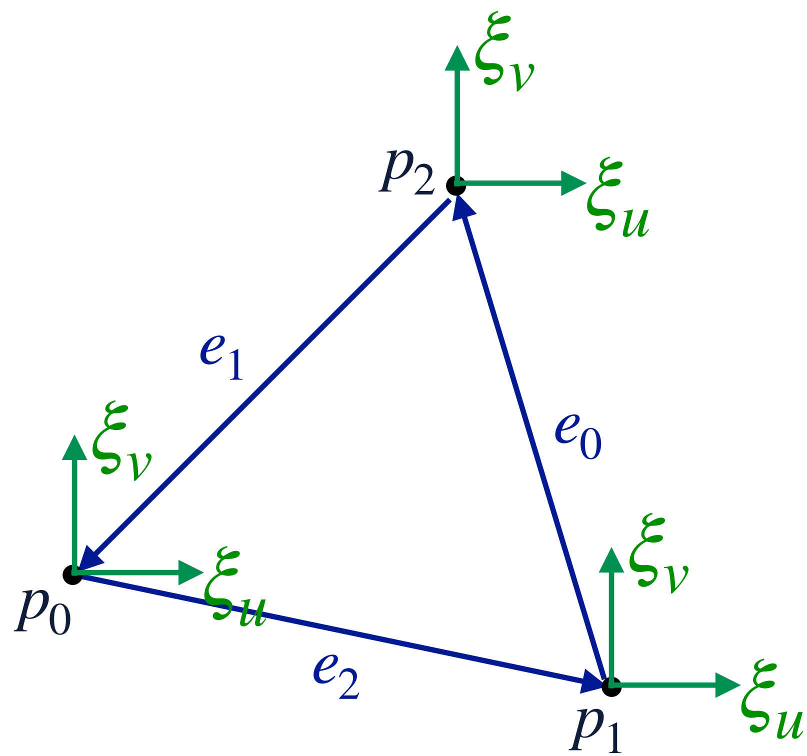
$$Df^T\left(DN\begin{bmatrix}u\\v\end{bmatrix}\right)\approx Df^T\Delta\vec{n} \implies S\begin{bmatrix}u\\v\end{bmatrix}\approx Df^T\Delta\vec{n}$$

$\because\ Df\begin{bmatrix}u\\v\end{bmatrix}=Y\in\mathbf{T}(\mathbb{R}^3)$ and $Df=[\vec{\xi}_u,\vec{\xi}_v]$ $\quad\therefore\begin{bmatrix}u\\v\end{bmatrix}=Df^TY$

$$\therefore\ S[Df]^TY\approx[Df]^T\Delta\vec{n}$$

$$\begin{cases} S[Df]^Te_0 = Df^T(\vec{n}_2-\vec{n}_1),\\ S[Df]^Te_1 = Df^T(\vec{n}_0-\vec{n}_2),\\ S[Df]^Te_2 = Df^T(\vec{n}_1-\vec{n}_0),\end{cases}$$

So we can solve $S\in\mathbb{R}^{2\times2}$ by least square(6 equations and 4 unknowns)



Summary of Mesh Curvature Estimation

- Rusinkiewicz's method is an effective approach for face curvature estimation
  - Szymon Rusinkiewicz, "Estimating Curvatures and Their Derivatives on Triangle Meshes", 3DPVT, 2004
- Good robustness to moderate amount of noise and free of degenerate configurations
- Can be used to compute curvatures for point cloud as well

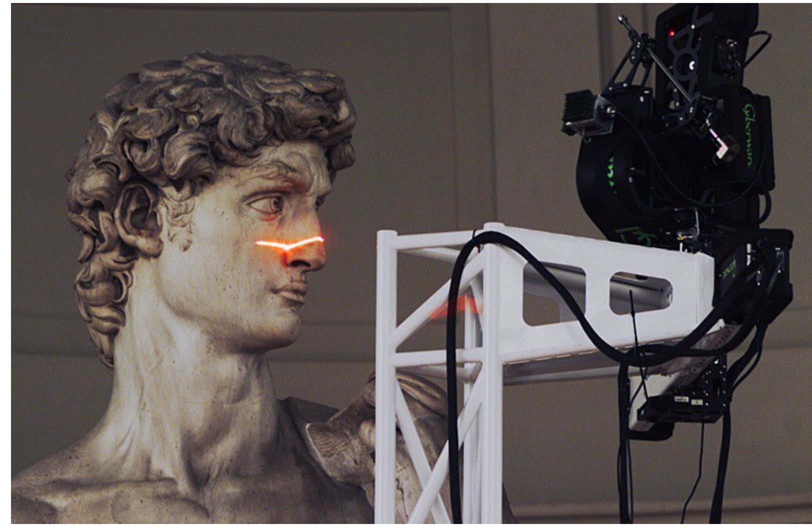
Point Cloud

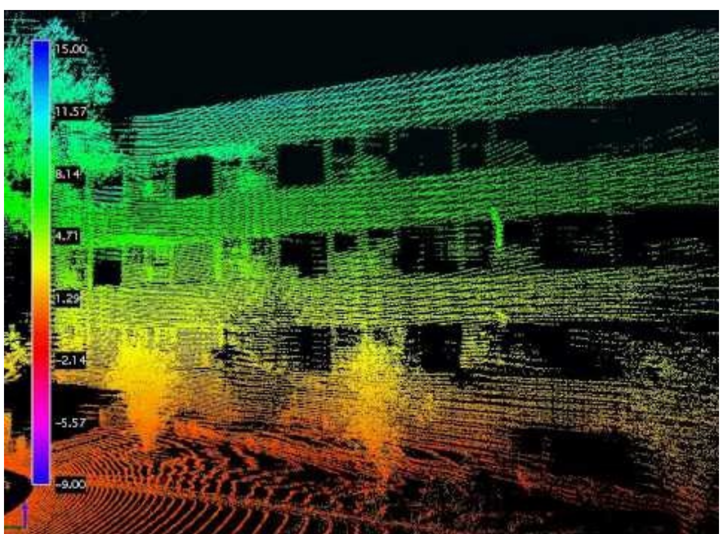
- Representation
- Sampling Points on Surfaces
- Normal Computation

Ack: Sid Chaudhuri

Acquiring Point Clouds

- From the real world
  - 3D scanning
    - Data is “striped”
    - Need multiple views to compensate occlusion



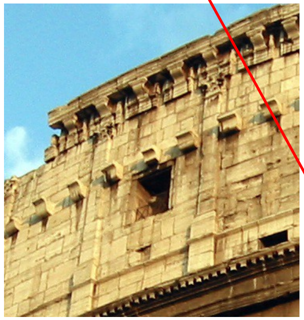


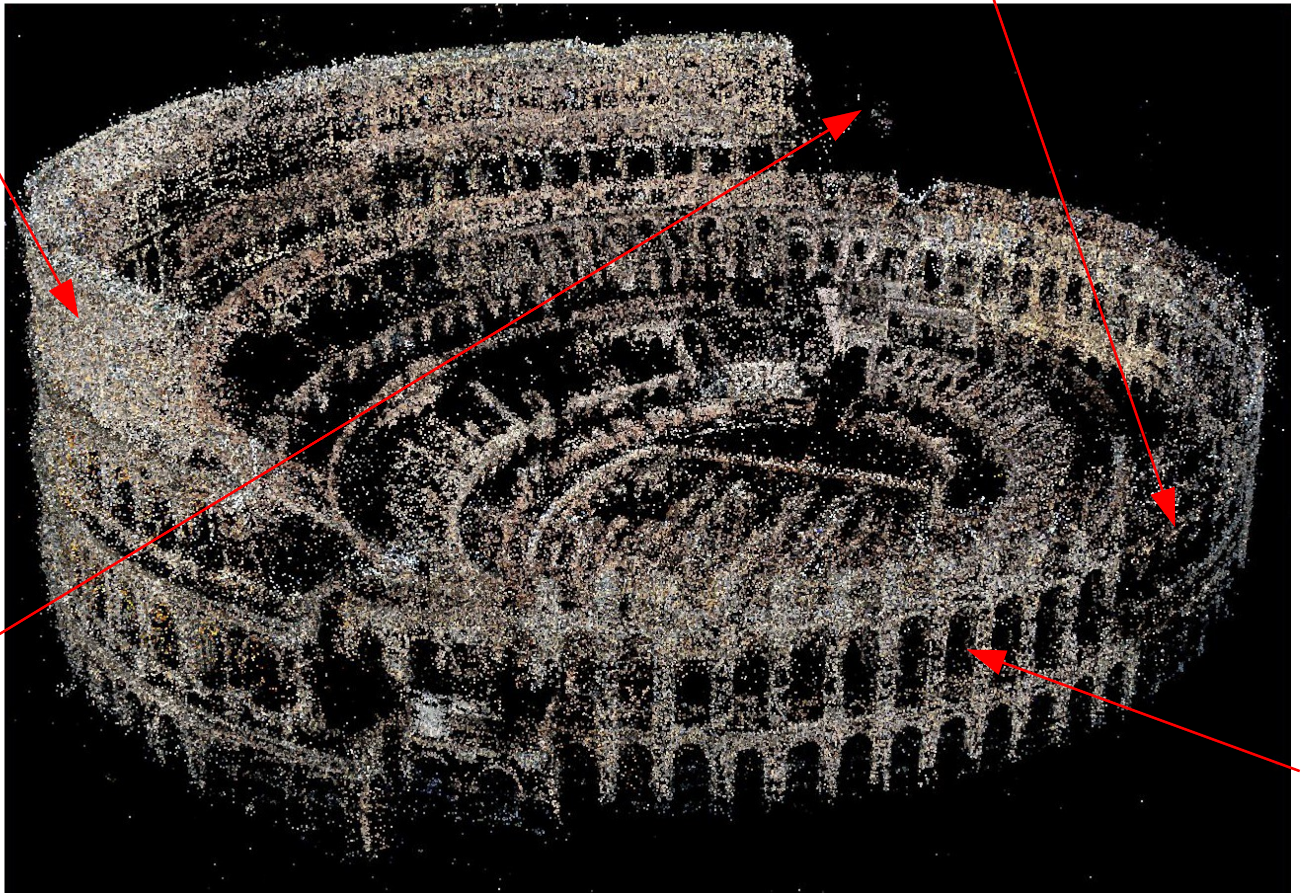
- Many techniques
  - Laser (LIDAR, e.g., StreetView)
  - Infrared (e.g., Kinect)
  - Stereo (e.g., Bundler)
- Many challenges: resolution, occlusion, noise, registration

Acquisition Challenges

Noise→Poor detail reproduction

Low resolution further obscures detail





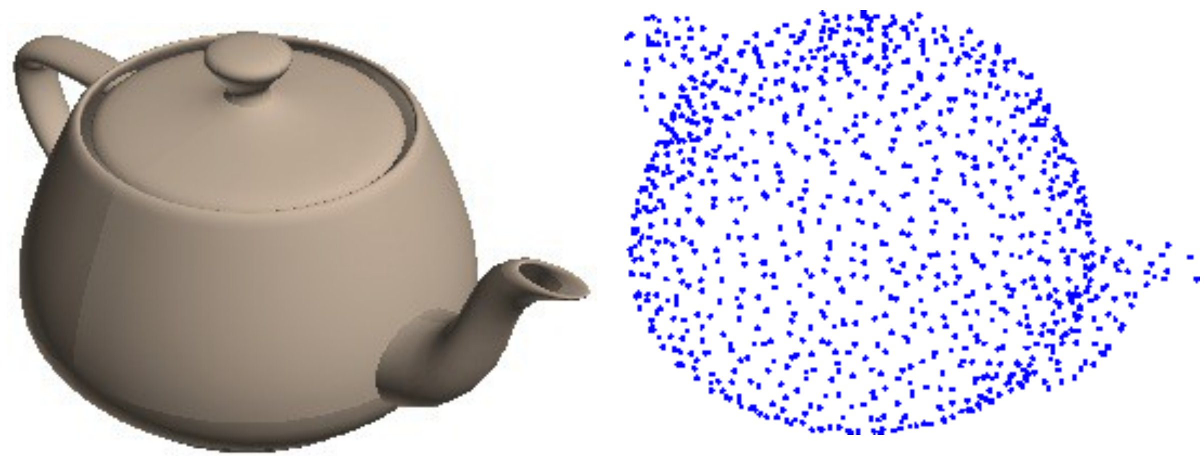
Some data was not properly registered with the rest

Occlusion→ Interiors not captured

http://grail.cs.washington.edu/

Acquiring Point Clouds

- From existing virtual shapes



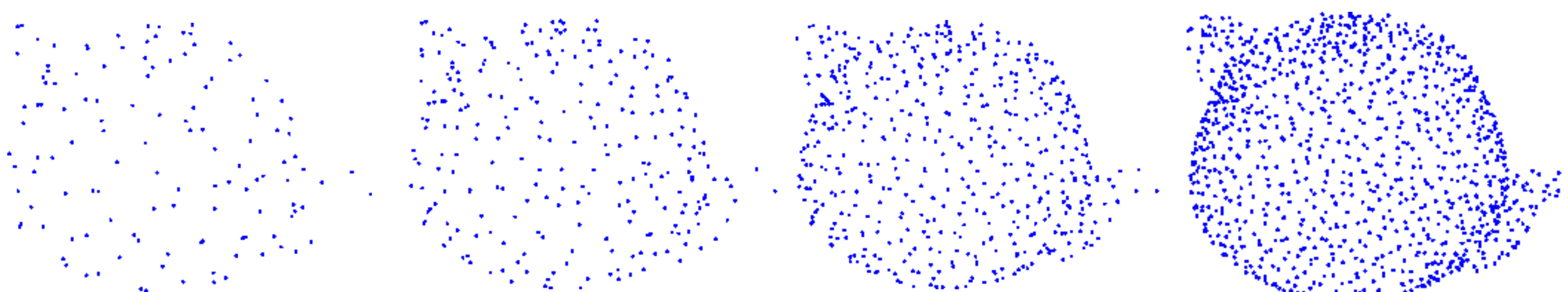
- Why would we want to do this?

Light-weight Shape Representation

Point cloud:

- Simple to understand
- Compact to store
- Generally easy to build algorithms

Yet already carries rich information!



$N = 125$ $N = 250$ $N = 500$ $N = 1000$

Point Cloud

- Representation
- Sampling Points on Surfaces
- Normal Computation

Ack: Sid Chaudhuri

Application-based Sampling

- For storage or analysis purposes (e.g., shape retrieval, signature extraction),
  - the objective is often to preserve surface information as much as possible
- For learning data generation purposes (e.g., sim2real),
  - the objective is often to minimize virtual-real domain gap

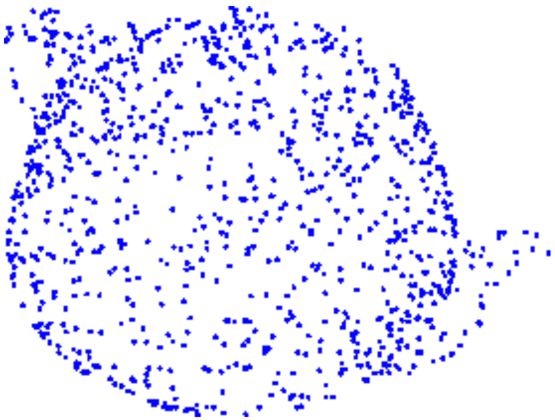
Application-based Sampling

- For storage or analysis purposes (e.g., shape retrieval, signature extraction),
  - the objective is often to preserve surface information as much as possible

- For learning data generation purposes (e.g., sim2real),
  - the objective is often to minimize virtual-real domain gap

Naive Strategy: Uniform Sampling

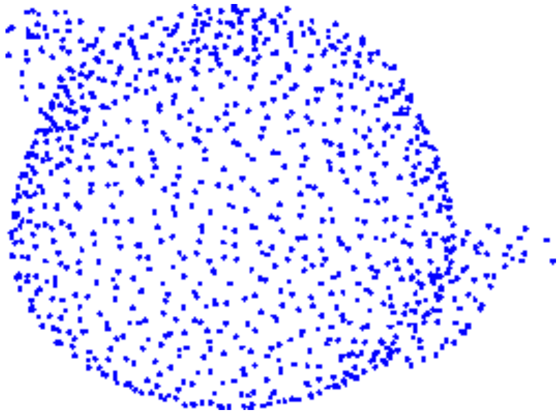
- Independent identically distributed (i.i.d.) samples by surface area:



- Usually the easiest to implement (as in your HW0)
- Issue: Irregularly spaced sampling

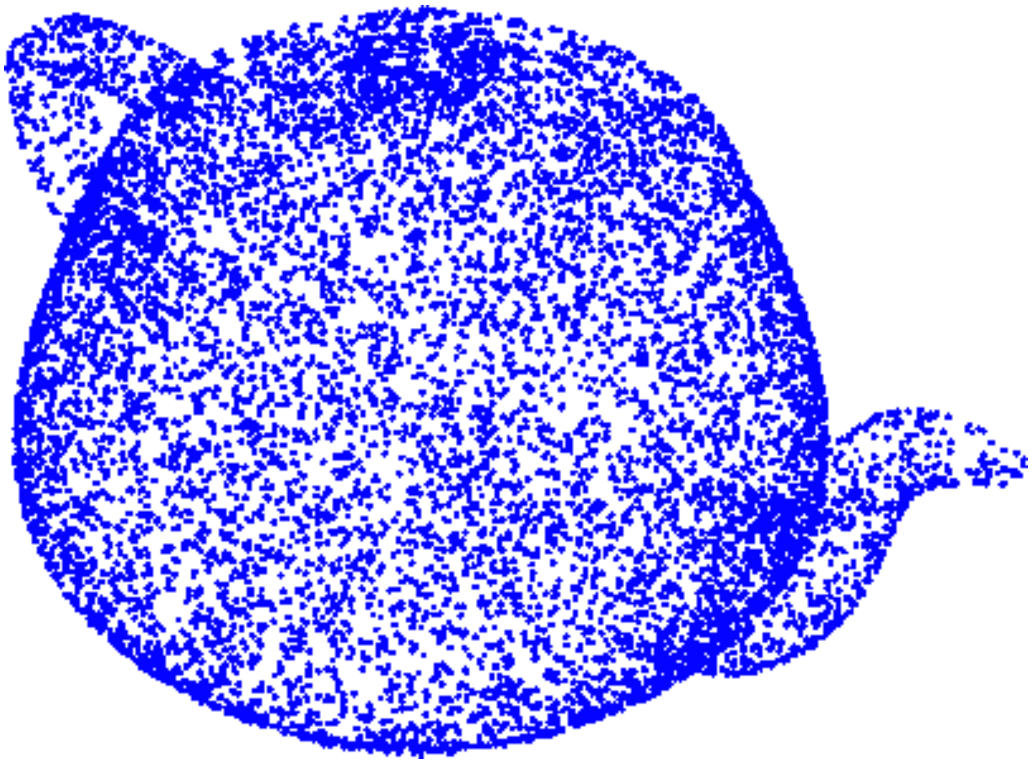
Farthest Point Sampling

- Goal: Sampled points are far away from each other
- NP-hard problem
- What is a greedy approximation method?



Iterative Furthest Point Sampling

- Step 1: Over sample the shape by any fast method (e.g., uniformly sample N=10,000 i.i.d. samples)



Iterative Furthest Point Sampling

- Step 2: Iteratively select K points

U is the initial big set of points
$S = \{\}$

add a random point from U to S

for i=1 to K
    find a point $u \in U$ with the largest distance to S
    add u to S

Issues Relevant to Speed

- Theoretically, naive implementation gives $\mathcal{O}(KN)$, but how to improve from $\mathcal{O}(KN)$ is an open question

- **Implementation can cause large speed difference**
  - As this is a serial algorithm in *K*, engineers optimize the efficiency in *N* (computing point-set distance)
  - CPU: Suggest using vectorization (e.g., numpy, scipy.spatial.distance.cdist)
  - GPU: By using shared memory, the complexity can be reduced to $\mathcal{O}(K(N/M + \log M))$, where *M* is the number of threads (*M=512* in practice for modern GPU).

Read by yourself!

Implementation Tricks

- References:
  - https://github.com/maxjaritz/mvpnet/blob/master/mvpnet/ops/cuda/fps_kernel.cu
  - https://github.com/erikwijmans/Pointnet2_PyTorch/blob/master/pointnet2_ops_lib/pointnet2_ops/_ext-src/src/sampling_gpu.cu
- By courtesy of Jiayuan Gu, we share a GPU version code with you (through Piazza)

Read by yourself!

An Implementation in Numpy

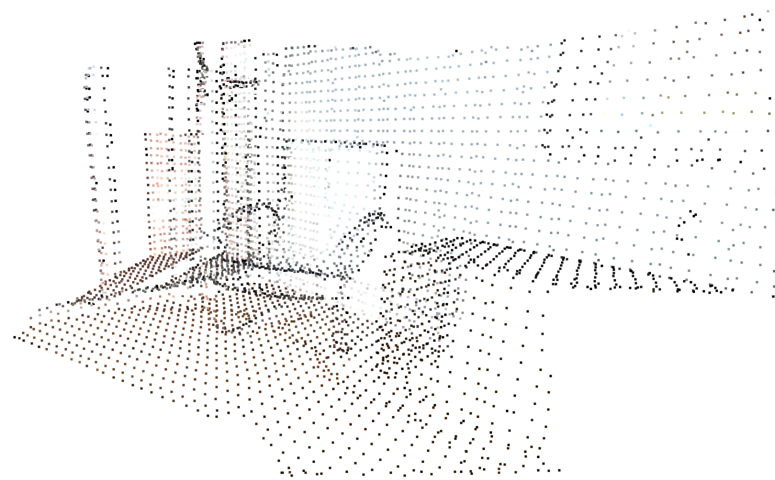
```
def fps_downsample(points, number_of_points_to_sample):
    selected_points = np.zeros((number_of_points_to_sample, 3))
    dist = np.ones(points.shape[0]) * np.inf # distance to the selected set
    for i in range(number_of_points_to_sample):
      # pick the point with max dist
      idx = np.argmax(dist)
      selected_points[i] = points[idx]
      dist_ = ((points - selected_points[i]) ** 2).sum(-1)
      dist = np.minimum(dist, dist_)

   return selected_points
```

Read by yourself!

Voxel Downsampling

- Uses a ***regular*** voxel grid to downsample, taking one point per grid
- Allows higher parallelization level
- Generates regularly spaced sampling (with noticeable artifacts)



Credit to: Open3D

Issues Relevant to Speed

- Need to map each point to a bin. Often implemented as adding elements into a hash table

- $\mathcal{O}(N)$ (assuming that the inserting into hash table takes O(1))

- On GPUs, parallelization reduces complexity of
  - Mapping each point to an integer value
  - Assign each value to an index so that the same value shares the same index
  - Aggregate indexes and form the output (called scattering in CUDA)

Read by yourself!

A Dictionary-based Implementation in Numpy

```
def voxel_downsample(points: np.ndarray, voxel_size: float):
    """Voxel downsample (first).

    Args:
        points: [N, 3]
        voxel_size: scalar

    Returns:
        np.ndarray: [M, 3]
    """
    points_downsampled = dict()  # point in each voxel cell
    points_voxel_coords = (points / voxel_size).astype(int)  # discretize to voxel
coordinate
    for point_idx, voxel_coord in enumerate(points_voxel_coords):
        key = tuple(voxel_coord.tolist())  # voxel coordinate
        if key not in points_downsampled:
            # assign the point to a voxel cell
            points_downsampled[key] = points[point_idx]
    points_downsampled = np.array(list(points_downsampled.values()))
    return points_downsampled
```

Read by yourself!

A Unique-based Implementation in Torch

```python
def voxel_downsample_torch(points: torch.Tensor, voxel_size: float):
    """Voxel downsample (average).

    Args:
        points: [N, 3]
        voxel_size: scalar

    Returns:
        torch.Tensor: [M, 3]
    """
    points = torch.as_tensor(points, dtype=torch.float32)
    points_voxel_coords = (points / voxel_size).long()  # discretize

    # Generate the assignment between points and voxel cells
    unique_voxel_coords, points_voxel_indices, count_voxel_coords = torch.unique(
        points_voxel_coords, return_inverse=True, return_counts=True, dim=0)

    M = unique_voxel_coords.size(0)  # the number of voxel cells
    points_downsampled = points.new_zeros([M, 3])
    points_downsampled.scatter_add_(
        dim=0,
        index=points_voxel_indices.unsqueeze(-1).expand(-1, 3),
        src=points)
    points_downsampled = points_downsampled / count_voxel_coords.unsqueeze(-1)
    return points_downsampled
```

Read by yourself!

Application-based Sampling

- For storage or analysis purposes (e.g., shape retrieval, signature extraction),
  - the objective is often to preserve surface information as much as possible

- For learning data generation purposes (e.g., sim2real),
  - the objective is often to minimize virtual-real domain gap
  - **a good research topic (e.g., GAN? Adversarial training? Differentiable sampling?)**

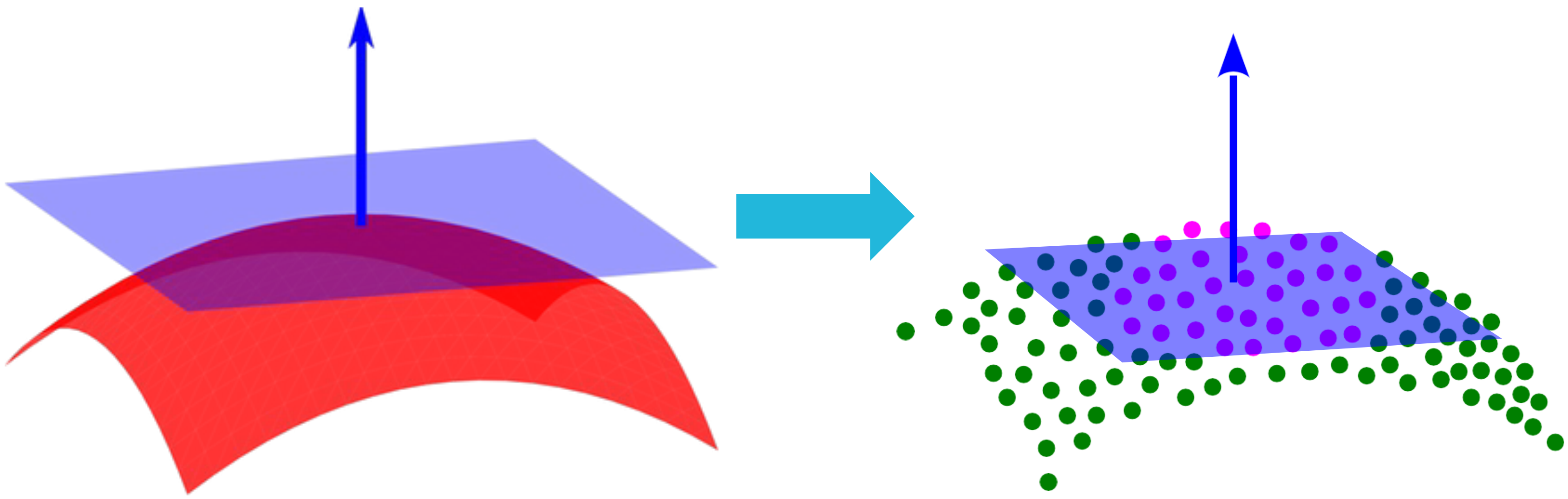
Point Cloud

- Representation
- Sampling Points on Surfaces
- Normal Computation

Ack: Sid Chaudhuri

Estimating Normals

- Plane-fitting: find the plane that best fits the neighborhood of a point of interest



Least-square Formulation

- Assume the plane equation is:

$$w^T(x - c) = 0 \quad \text{with} \quad \|w\| = 1$$

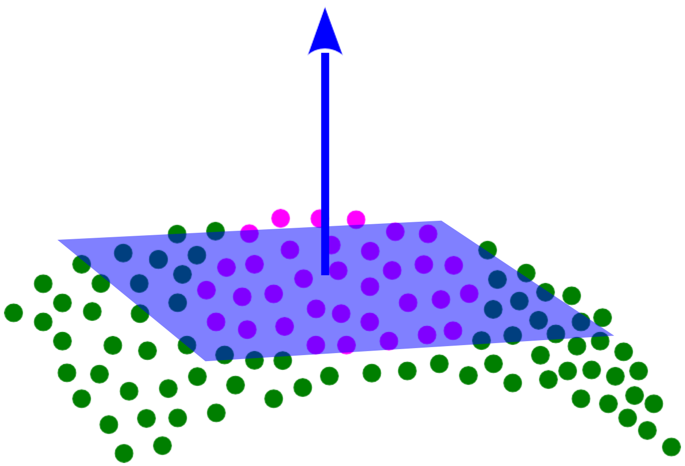
- Plane-fitting solves the least square problem:

$$\text{minimize}_{w,c} \quad \sum_i \|w^T(x_i - c)\|_2^2$$

$$\text{subject to} \quad \|w\|^2 = 1$$

where $\{x_i\}$ is the neighborhood of a point x that you query the normal

- Doing Lagrangian multiplier and the solution is:
  - Let $M = \sum_i (x_i - \bar{x})(x_i - \bar{x})^T$ and $\bar{x} = \frac{1}{n}\sum_i x_i$,
  - w: the smallest eigenvector of M
  - $c = w^T\bar{x}$



- w also corresponds to the third principal component of M (yet another usage of PCA)
  - Where are the first and second principal components?

Summary of Normal Computation

- The normal of a point cloud can be computed through PCA over a local neighborhood

- Remark:
    - The choice of neighborhood size is important
    - When outlier points exist, RANSAC can improve quality