# Digital Geometry Processing

Algorithms for Representing, Analyzing and Comparing 3D shapes

# Announcements

- Next week is Midterm 2 (on lectures 4 – 6)
- Project Presentations
  - December 17th-19th (Monday – Wednesday)
  - We will send a Doodle
- Pick your projects!

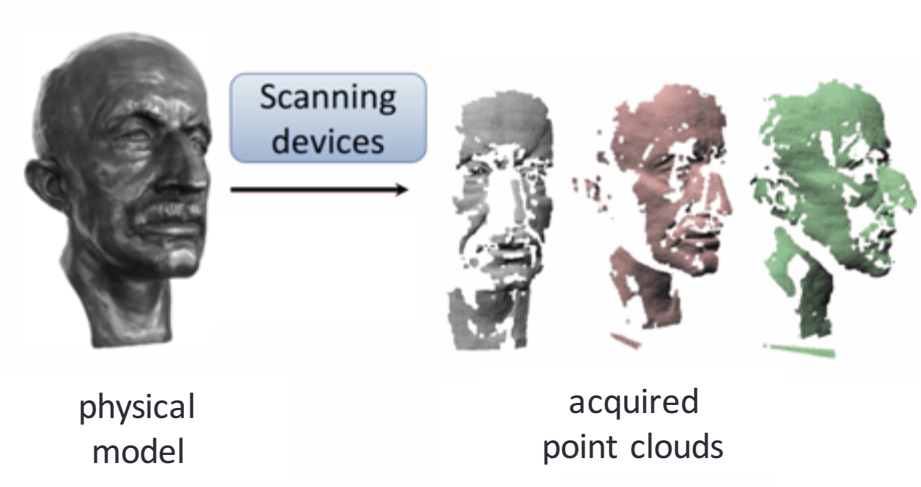
# Today

- Point Cloud Analysis Recap
- Shape Reconstruction (continued)
- Subdivision Curves
- Subdivision Surfaces

# Last Time


Scanning devices
Registration
Implicit reconstruction
physical model
acquired point clouds
registered point cloud
reconstructed model

# Last Time




Types of 3D scanners:

- Time of Flight
  - Delay-based
  - Frequency-based
- Triangulation
  - Laser-based (single line)
  - Structured light (multiple patterns)
- Computer Vision-based
  - Depth-from-stereo
  - Depth-from-blur
- Example: Microsoft Kinect

# Last Time


acquired
point clouds

Partial Scans -> Single Point Cloud

Main Approach: Iterative Closest Point.

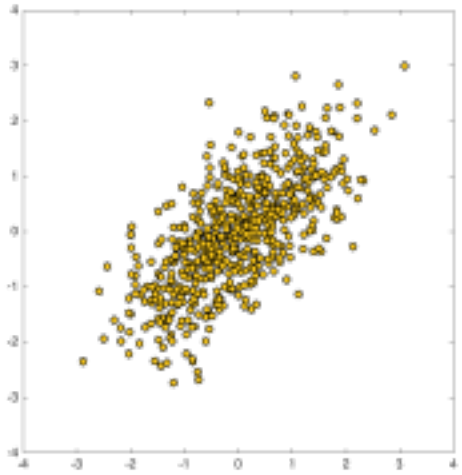
At each iteration:

1) Find nearest neighbor
2) Find best transformation
    a. Point-to-point (closed form solution)
    b. Point-to-plane (local linearization)

# Principal Component Analysis

PCA more generally (works in any dimension):

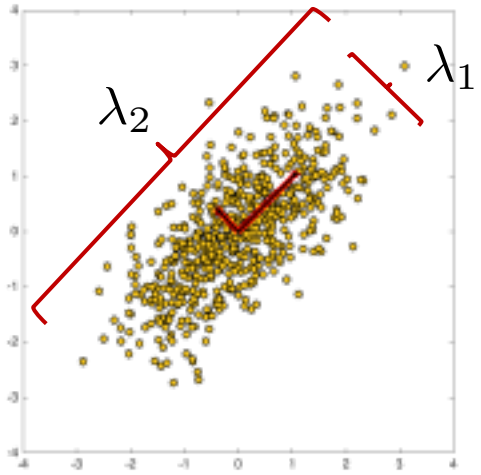
1. Given a point cloud with $n$ points in $\mathbb{R}^d$
2. Covariance matrix $C \in \mathbb{R}^{d \times d}$, $C = \sum_{i=1}^{n} (p_i - \mu)(p_i - \mu)^T$
3. Eigenvectors of $C$ encode *principal directions*
4. Eigenvalues (all non-negative!) encode the *variance.*



# Principal Component Analysis

PCA more generally (works in any dimension):

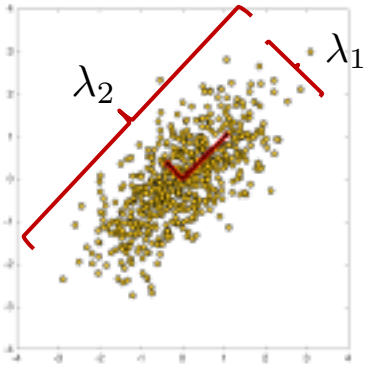
1. Given a point cloud with $n$ points in $\mathbb{R}^d$
2. Covariance matrix $C \in \mathbb{R}^{d \times d}$, $C = \sum_{i=1}^{n} (p_i - \mu)(p_i - \mu)^T$
3. Eigenvectors of $C$ encode *principal directions*
4. Eigenvalues (all non-negative!) encode the *variance.*



# Principal Component Analysis

PCA more generally (works in any dimension):

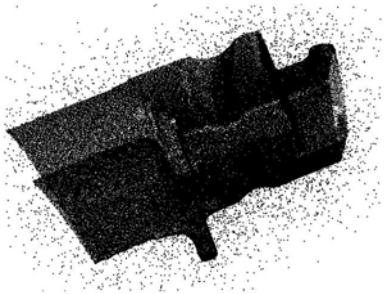
1. Given a point cloud with $n$ points in $\mathbb{R}^d$
2. Covariance matrix $C \in \mathbb{R}^{d \times d}$, $C = \sum_{i=1}^{n} (p_i - \mu)(p_i - \mu)^T$
3. Eigenvectors of $C$ encode *principal directions*
4. Eigenvalues (all non-negative!) encode the *variance.*
5. Ratios of eigenvalues can be used to distinguish e.g. flat regions.

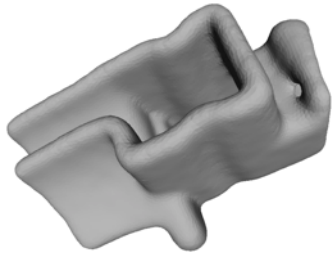


# 3d Shape Reconstruction Pipeline

Typically point cloud sampling of a shape is insufficient for most applications. Main stages in processing:

1. Shape scanning (acquisition)
2. If have multiple scans, align them.
3. Smoothing – remove local noise and outliers.
4. Estimate surface normals.
5. Surface reconstruction
   - Implicit representation
   - Triangle mesh

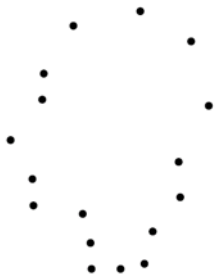


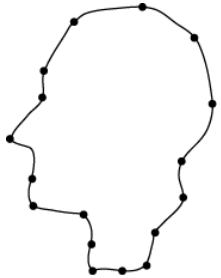


# 3d Point Cloud Reconstruction

Main Goal:

Construct a polygonal (e.g. triangle mesh) representation of the point cloud.



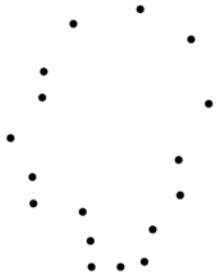


PCD → Reconstruction algorithm → curve/ surface

# 3d Point Cloud Reconstruction

Main Problem:

Data is **unstructured.** E.g. in 2D the points are not **ordered.**

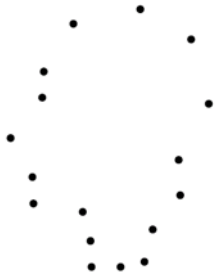


PCD → Reconstruction algorithm → curve/ surface

# 3d Point Cloud Reconstruction

Main Problem:

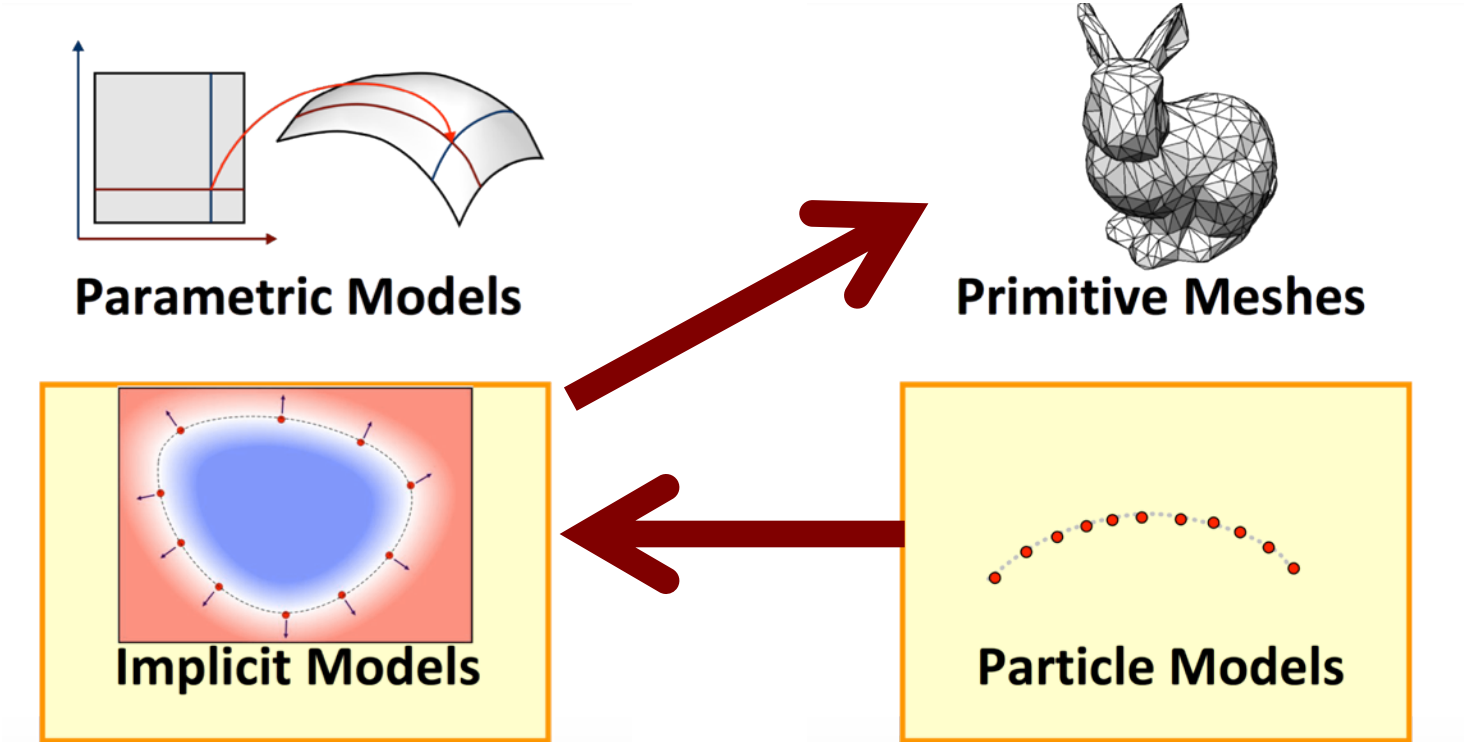
Data is **unstructured.** E.g. in 2D the points are not **ordered.**
Inherently **ill-posed** (aka difficult) problem.



PCD → Reconstruction algorithm → curve/ surface

# 3d Point Cloud Reconstruction

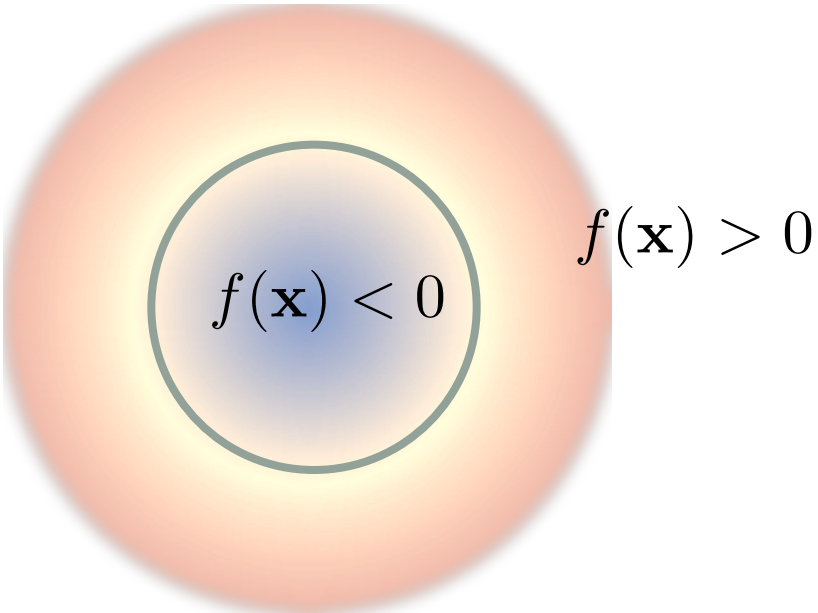
Today: Reconstruction through Implicit models.



# Implicit surfaces

Given a function f(**x**), the surface is defined as:

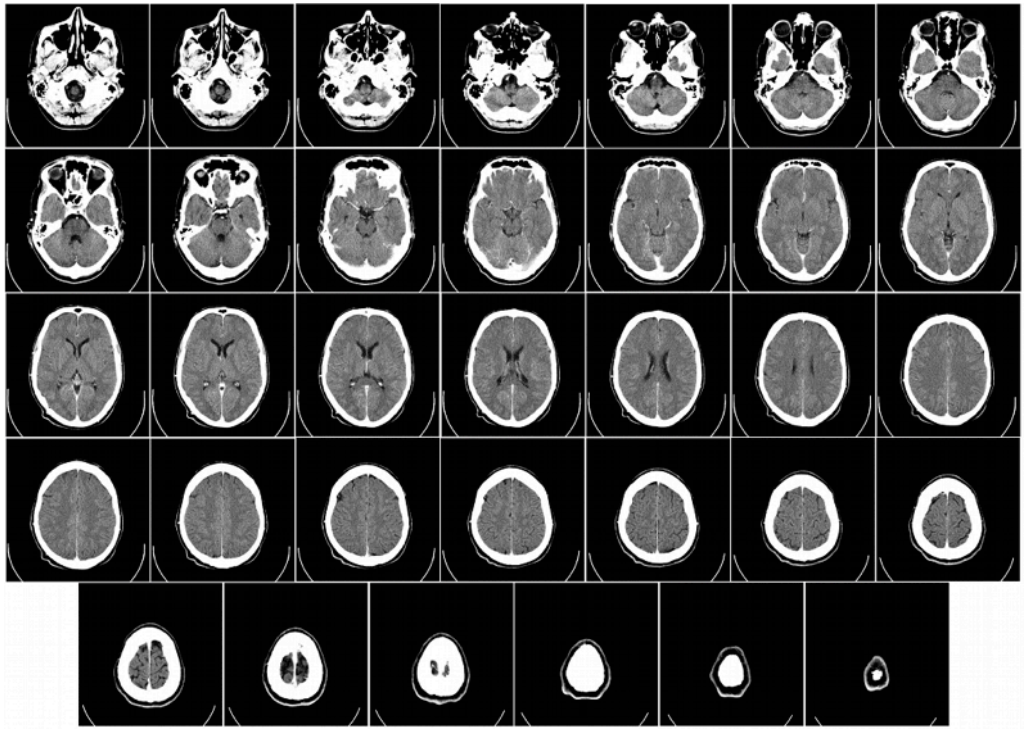
$$\{\mathbf{x}, \text{s.t.} f(\mathbf{x}) = 0\}$$




$$f(x, y) = x^2 + y^2 - r^2$$

# Implicit surfaces

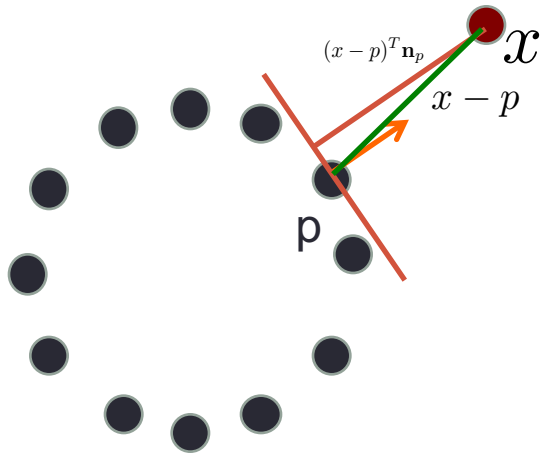
Some 3d scanning technologies (e.g. CT, MRI) naturally produce implicit representations



CT scans of human brain

# Implicit surfaces

Converting from a point cloud to an implicit surface:

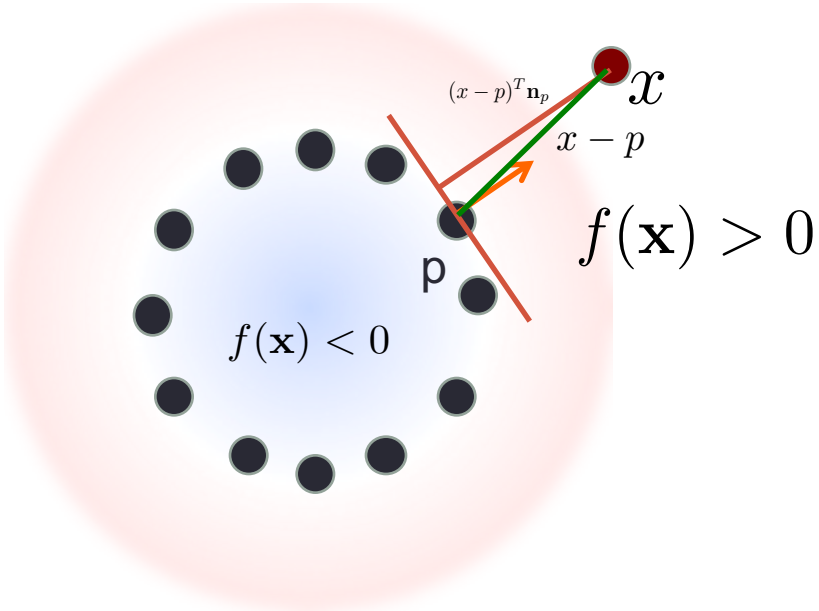



Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

# Implicit surfaces

Converting from a point cloud to an implicit surface:

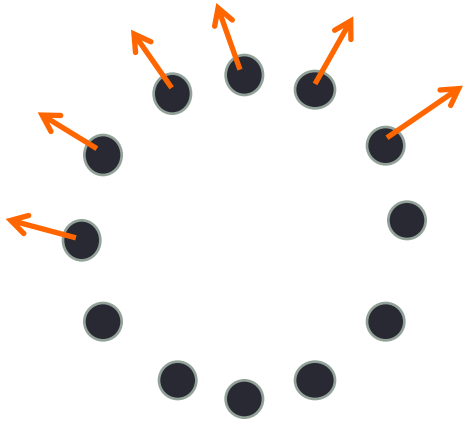



Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

# Implicit surfaces

Converting from a point cloud to an implicit surface:



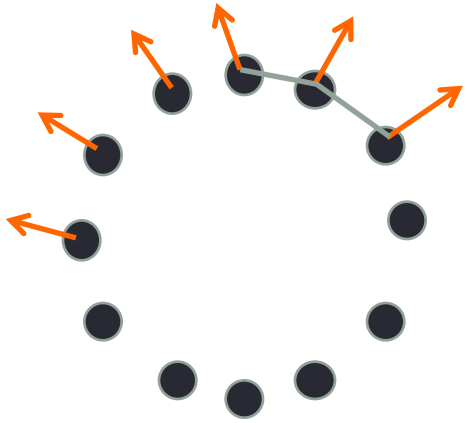
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x - p)^T \mathbf{n}_p$ – signed distance to the tangent plane.
3. Note: need consistently oriented normals.

   PCA only gives normals **up to orientation**

# Implicit surfaces

Converting from a point cloud to an implicit surface:



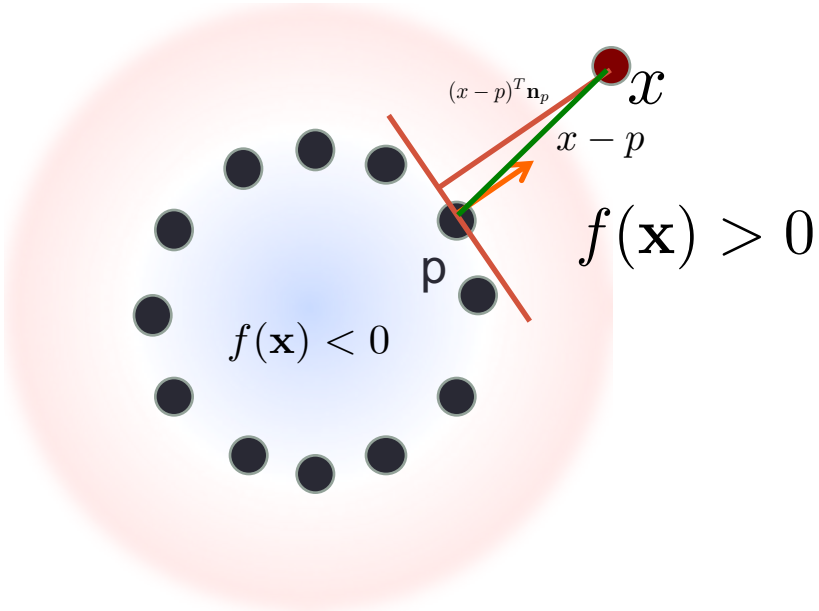
Simplest method:

1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x - p)^T \mathbf{n}_p$ – signed distance to the tangent plane.
3. Note: need consistently oriented normals. In general, difficult problem, but can try to locally connect points and fix orientations.

Hugues Hoppe: Surface reconstruction from unorganized points

# Implicit surfaces

Converting from a point cloud to an implicit surface:




Simplest method:

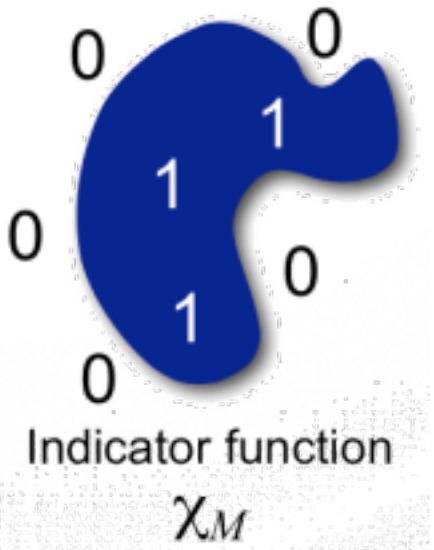
1. Given a point x in space, find nearest point p in PCD.
2. Set $f(x) = (x-p)^T \mathbf{n}_p$ – signed distance to the tangent plane.

**Note**: many more advanced methods exist:
e.g. Moving Least Squares (MLS)

# Poisson Surface Reconstruction

Main Goal: construct an indicator function of a surface

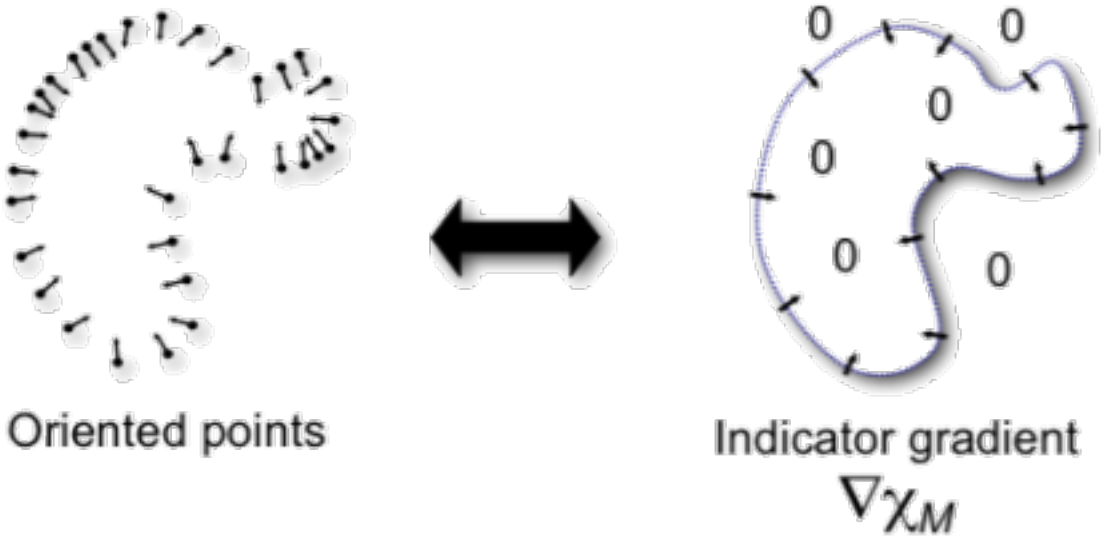
$$\chi_M(p) = \begin{cases} 1 & \text{if } p \in M \\ 0 & \text{if } p \notin M \end{cases}$$




M. Kazhdan, M. Bolitho, H. Hoppe *Poisson Surface Reconstruction,* Proc. SGP 2006

# Poisson Surface Reconstruction

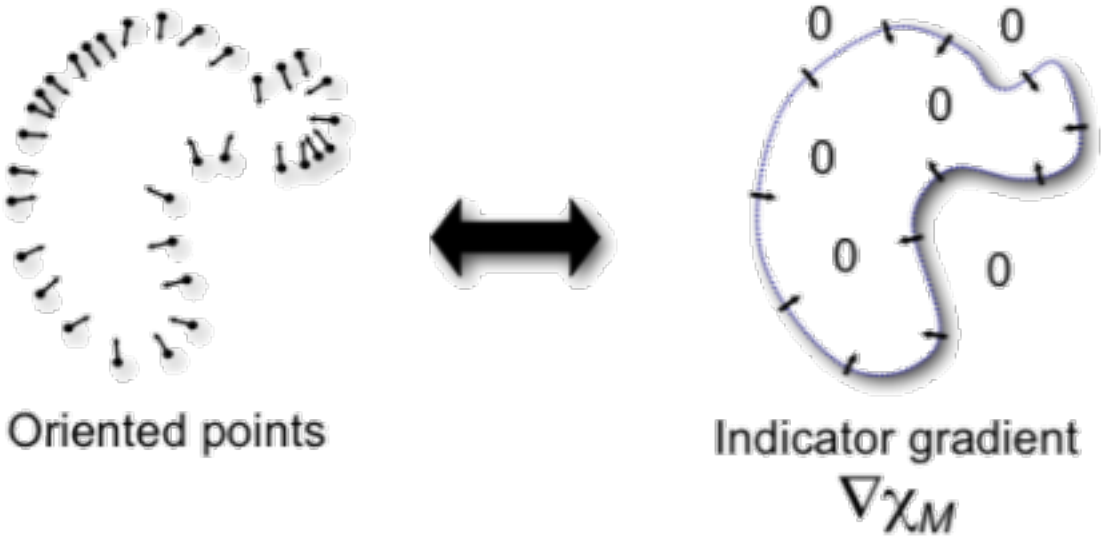
Observation:




The gradient of the indicator function should agree with the vector field of oriented points.

# Poisson Surface Reconstruction

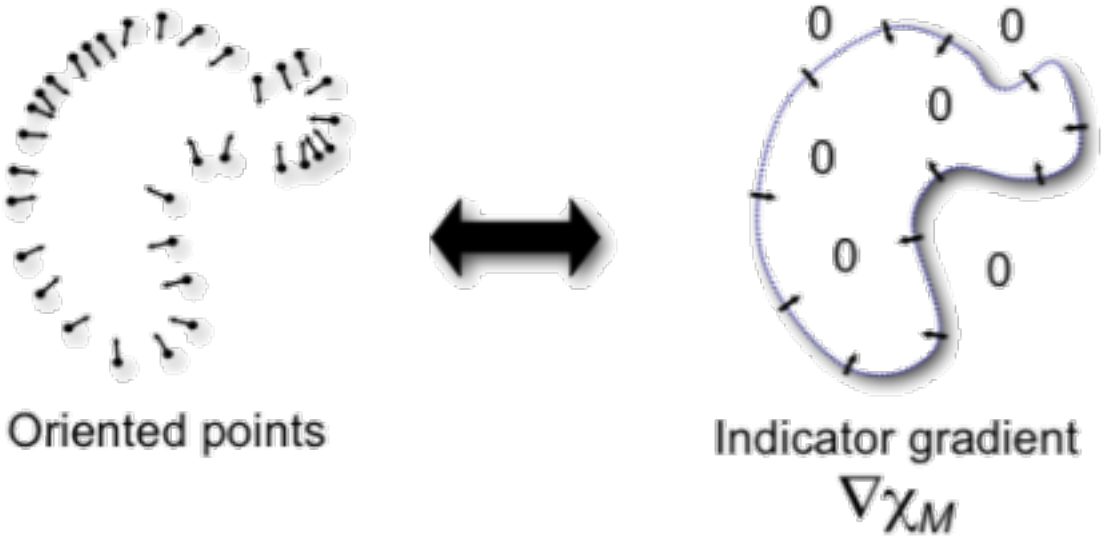
Solving for the indicator function:




$$\min_{\chi} \|\nabla\chi - V\|$$

# Poisson Surface Reconstruction

Solving for the indicator function:

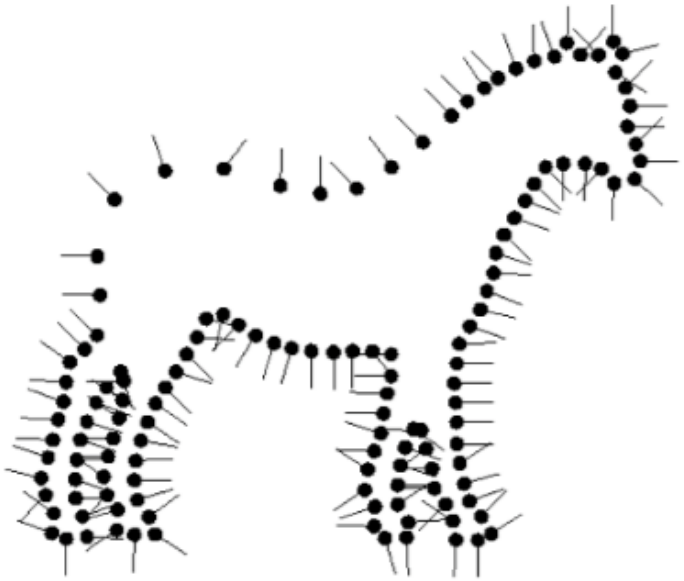



Or, applying the divergence operator:

$$\min_{\chi} \|\Delta\chi - \mathrm{div}(V)\|$$

# Poisson Surface Reconstruction

Given the indicator function – extract the isosurface.



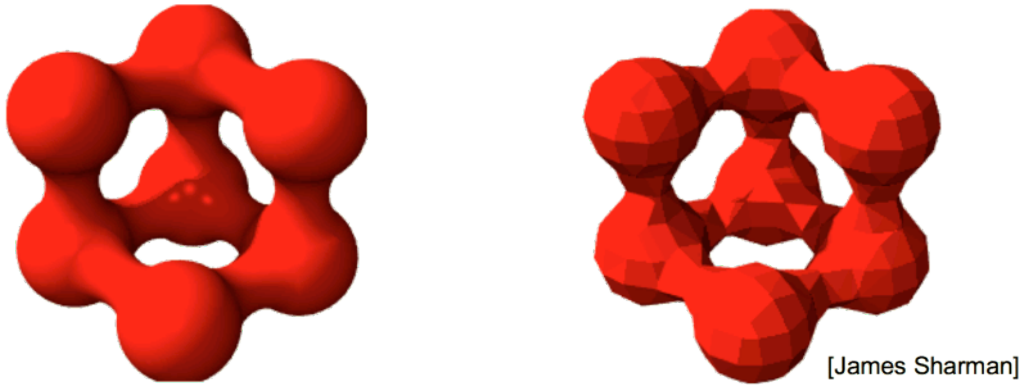
M. Kazhdan, M. Bolitho, H. Hoppe *Poisson Surface Reconstruction,* Proc. SGP 2006

# Marching Cubes

Converting from implicit to explicit representations.

Goal: Given an implicit representation: $\{\mathbf{x}, \mathrm{s.t.} f(\mathbf{x}) = 0\}$

Create a triangle mesh that approximates the surface.




Lorensen and Cline, SIGGRAPH ‘87

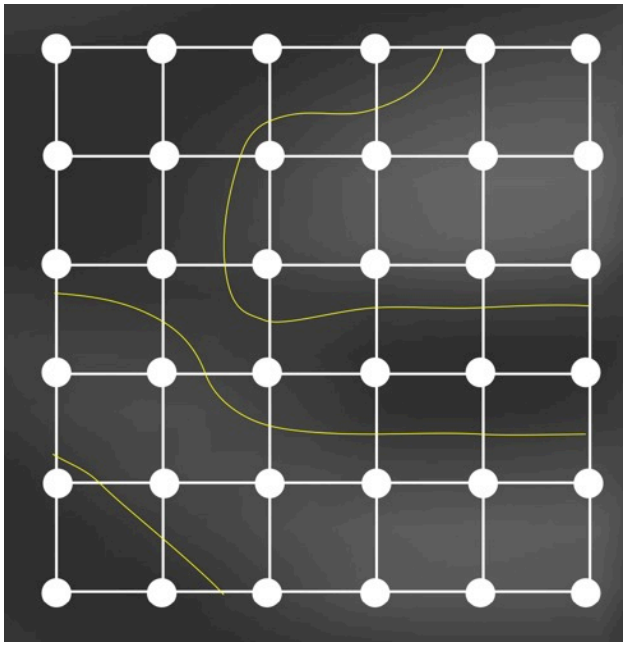
One of the most cited computer graphics papers of all time.

# Marching Squares (2D)

Given a function: $f(x)$

- $f(\mathbf{x}) < 0$ inside
- $f(\mathbf{x}) > 0$ outside

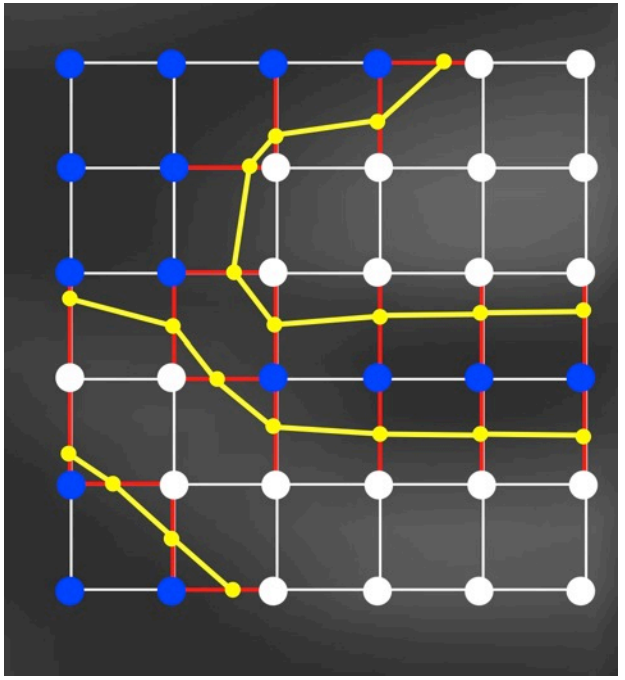
1. Discretize space.
2. Evaluate $f(x)$ on a grid.



# Marching Squares (2D)

Given a function: $f(x)$

- $f(\mathbf{x}) < 0$ inside
- $f(\mathbf{x}) > 0$ outside

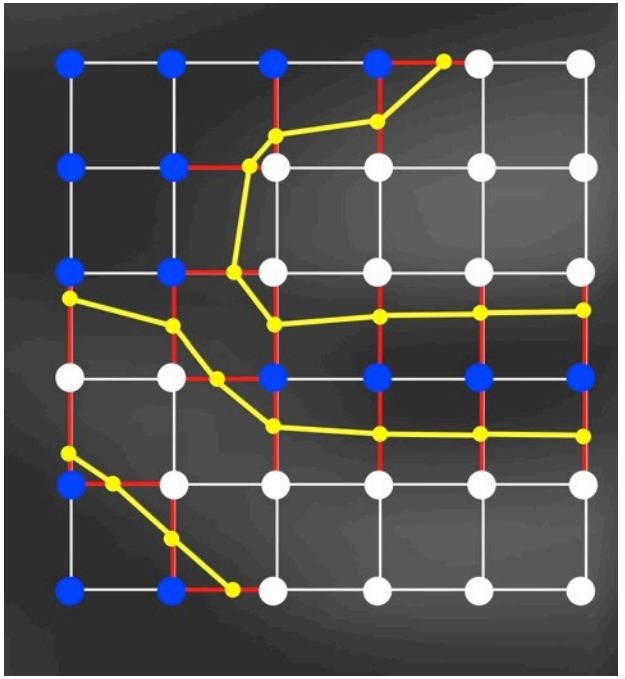


1. Discretize space.
2. Evaluate $f(x)$ on a grid.
3. Classify grid points (+-)
4. Classify grid edges
5. Compute Intersections
6. Connect Intersections

# Marching Squares (2D)

Connecting the intersections:

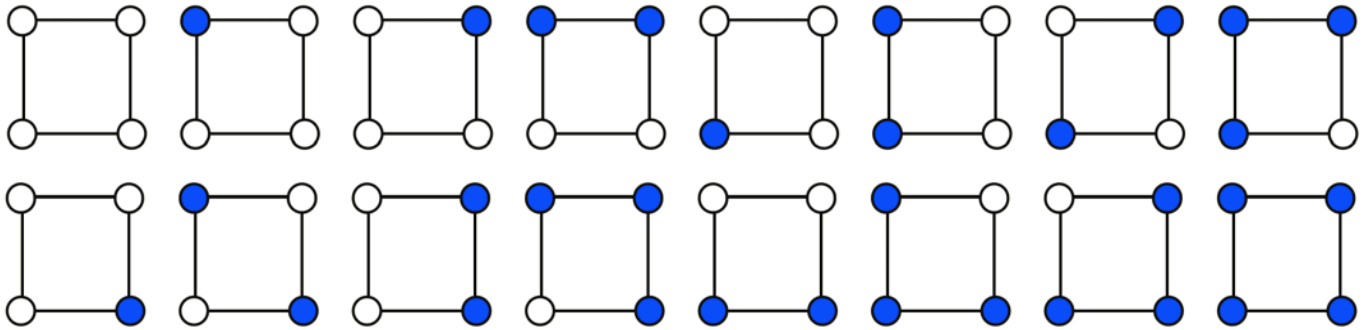
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.



# Marching Squares (2D)

Connecting the intersections:

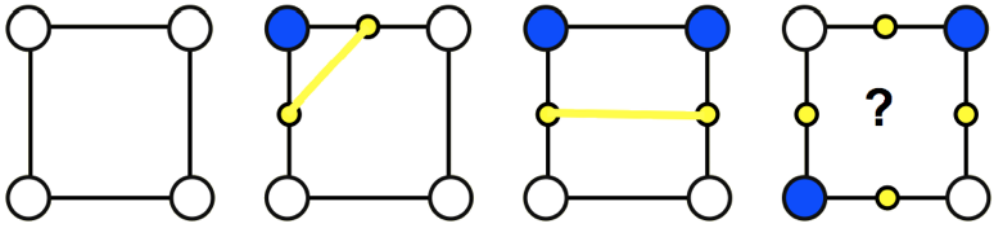
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections



# Marching Squares (2D)

Connecting the intersections:

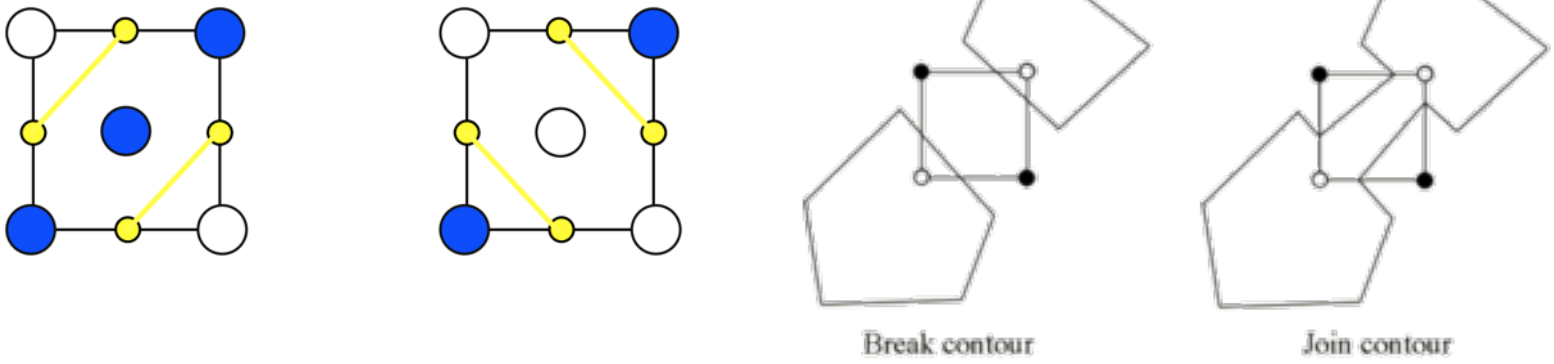
- Grand principle: treat each cell separately!
- Enumerate all possible inside/outside combinations.
- Group those leading to the same intersections.
- Group equivalent after rotation.
- Connect intersections



# Marching Squares (2D)

Connecting the intersections:

Ambiguous cases:




2 options:

1) Can resolve ambiguity by subsampling inside the cell.
2) If subsampling is impossible, pick one of the two possibilities.

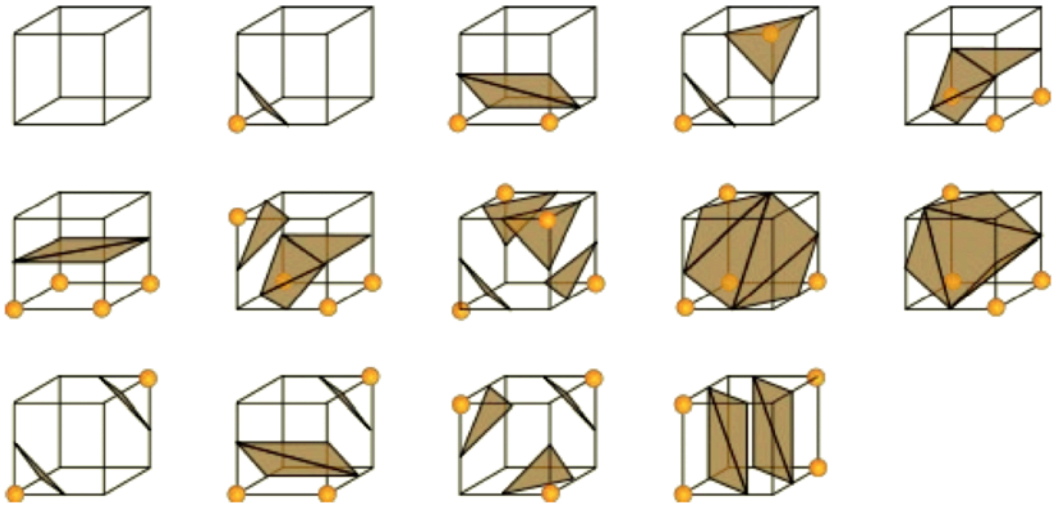
# Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 14 after symmetries



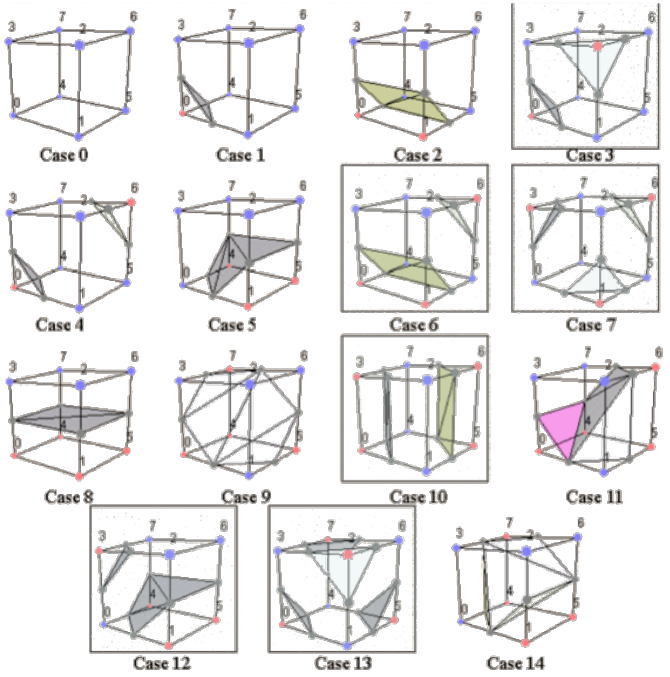
# Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 14 after symmetries
- 6 ambiguous cases (in boxes)



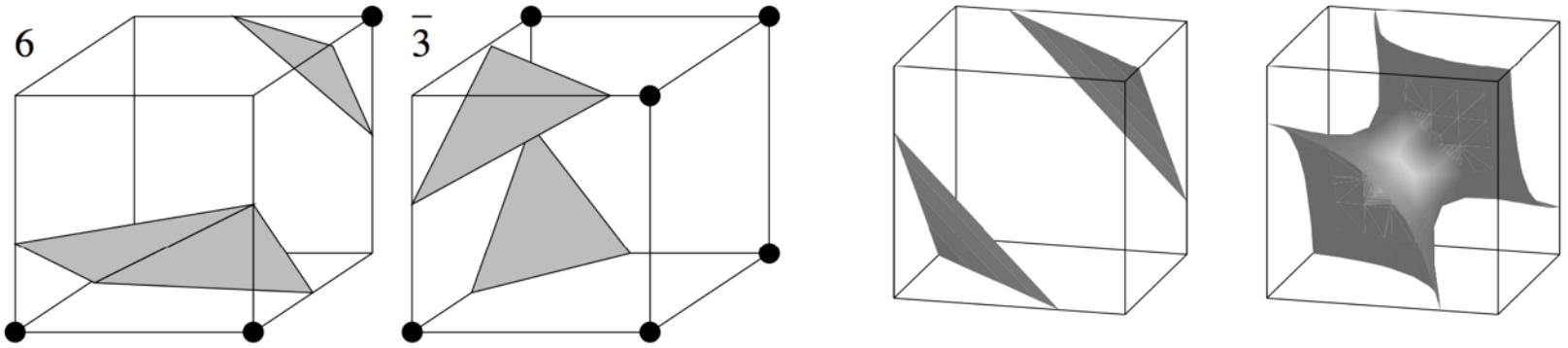
# Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 14 after symmetries
- 6 ambiguous cases (in boxes)
- Inconsistent triangulations can lead to holes and wrong topology.



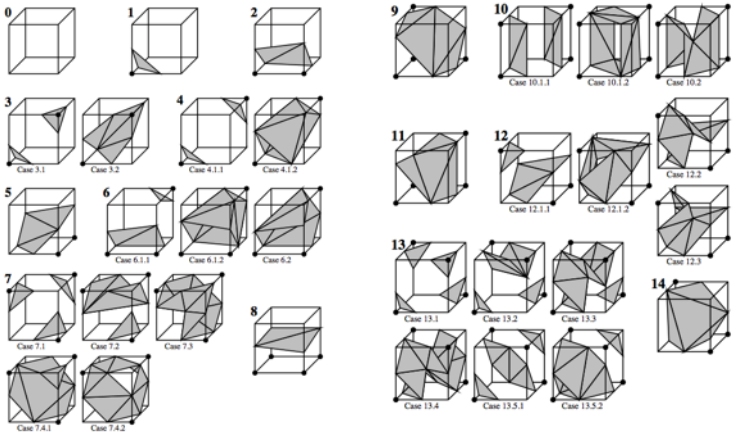
# Marching Cubes (3D)

Same basic machinery applies to 3d.

cells become **cubes** (voxels)

lines become **triangles**

- 256 different cases
- 14 after symmetries
- 6 ambiguous cases (in boxes)
- Inconsistent triangulations can lead to holes and wrong topology.
- More subsampling rules – leads to 33 unique cases.



Chernyaev, Marching Cubes 33,'95

# Marching Cubes (3D)

Main Strengths:

- Very multi-purpose.
- Extremely fast and parallelizable.
- Relatively simple to implement.
- Virtually parameter-free

Main Weaknesses:

- Can create badly shaped (skinny) triangles.
- Basic versions do not provide topological guarantees.
- Many special cases (implemented as big lookup tables).
- No sharp features.

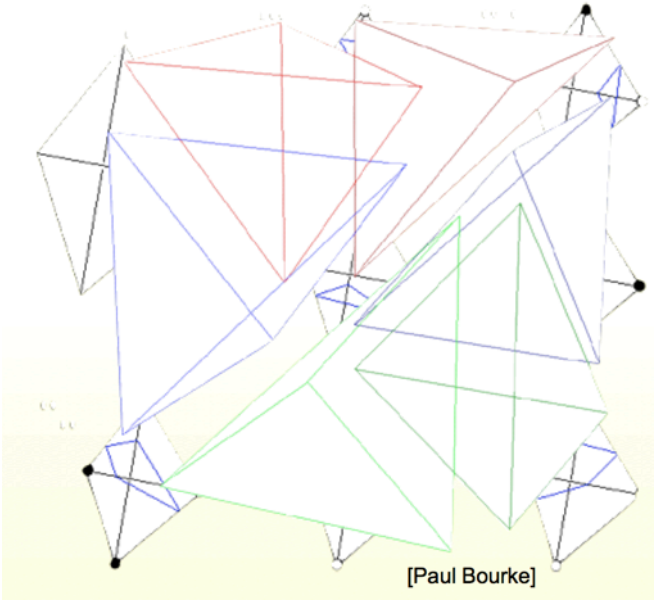
# Marching Cubes – Extensions

Marching Tetrahedra.

Instead of cubes (grid) use **Tetrahedra.**

- 6 Tetrahedra per voxel
- 16 cases (8 after symmetry)
- Up to 2 triangles per tet
- **No ambiguities.**

Can be used when input is discretized as tetrahedra.




Regularised marching tetrahedra: improved iso-surface extraction, G.M. Treece, R.W. Prager, A.H. Gee, Computers & Graphics, 1999.

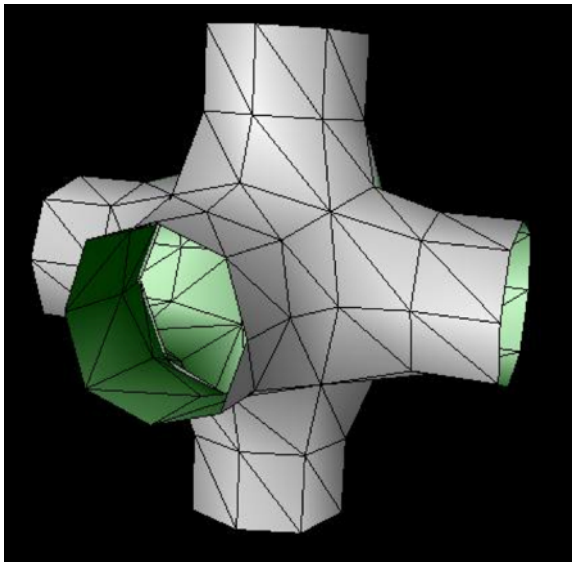
# Typical Scanning and Reconstruction Pipeline


**Questions?**

# Subdivision Surfaces

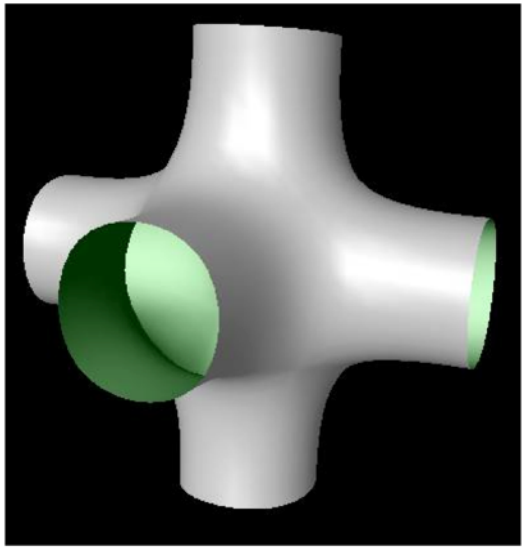
Provide a trade-off between Smooth and Mesh techniques:

- Inherently **continuous**
- Intuitive controls (control mesh)
- Can model shapes with arbitrary topology



Modeling

Subdivision surfaces

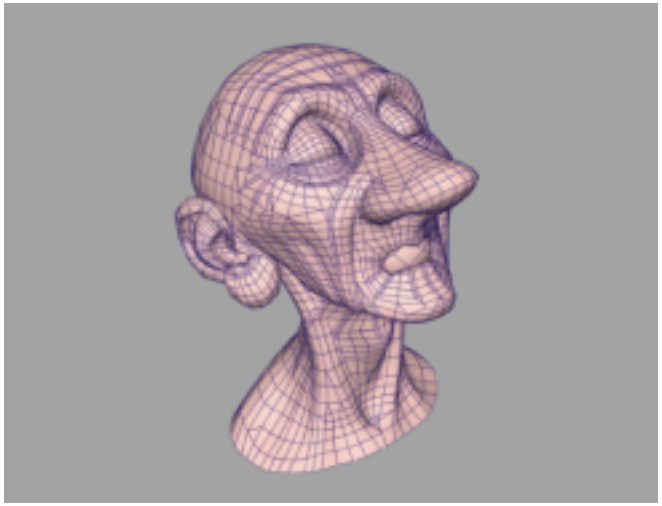


Rendering

# Subdivision Surfaces

Provide a trade-off between Smooth and Mesh techniques:

- Inherently **continuous**
- Intuitive controls (control mesh)
- Can model shapes with arbitrary topology



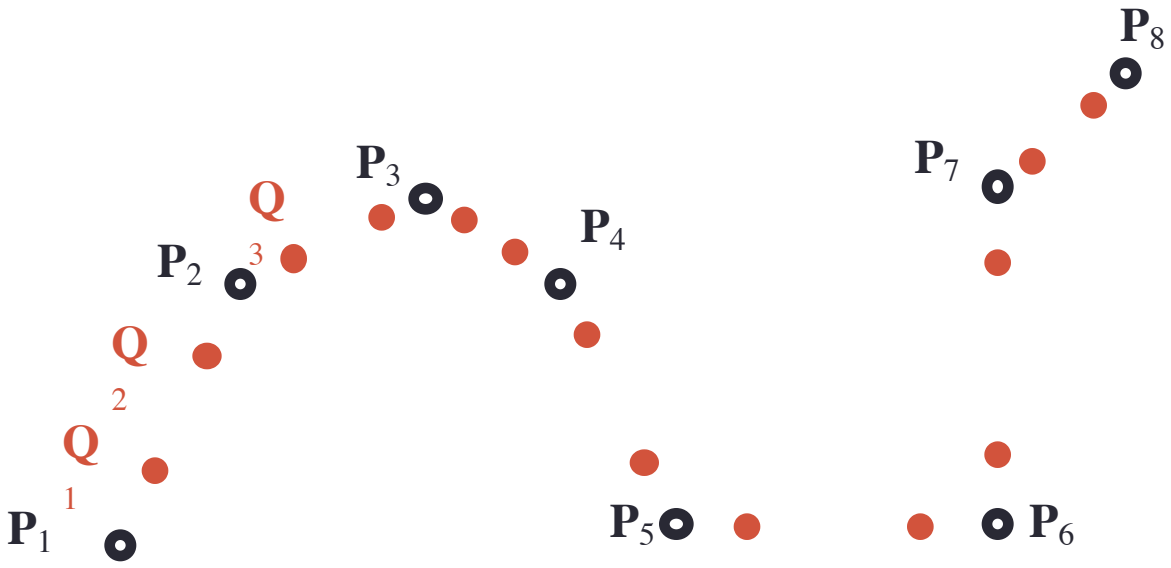
Modeling

Subdivision surfaces

Rendering

# Subdivision Curves

Uniform B-spline of order 2:



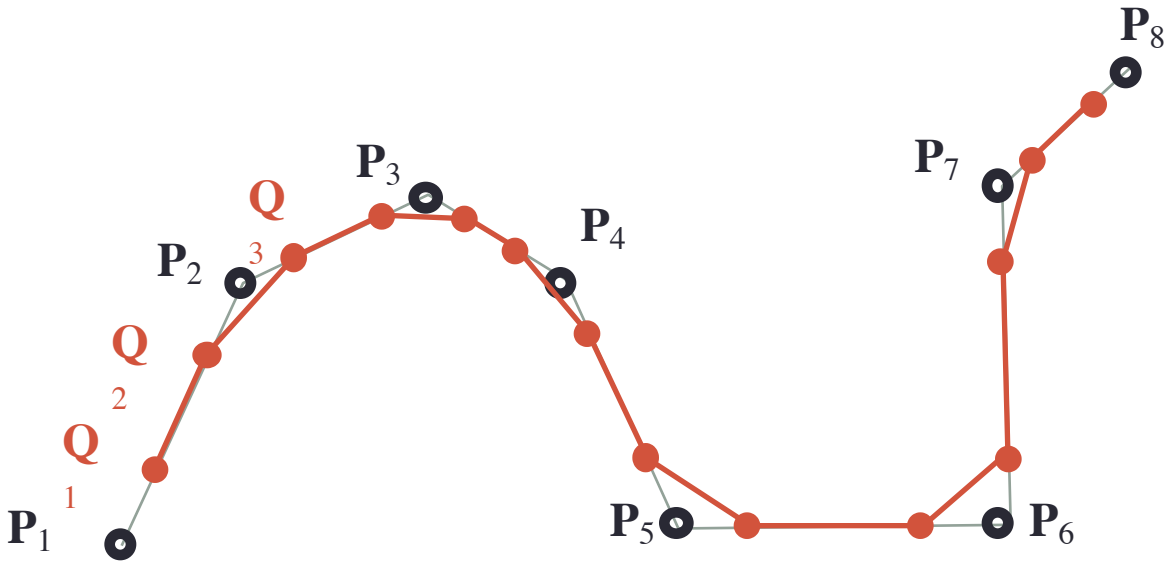

Chaikin's algorithm for Quadratic Uniform B-splines:

$$j \text{ odd: } \mathbf{Q}_j = \frac{3}{4}\mathbf{P}_{(j+1)/2} + \frac{1}{4}\mathbf{P}_{(j+3)/2}$$

$$j \text{ even: } \mathbf{Q}_j = \frac{1}{4}\mathbf{P}_{j/2} + \frac{3}{4}\mathbf{P}_{(j+2)/2}$$

# Subdivision Curves

Uniform B-spline of order 2:



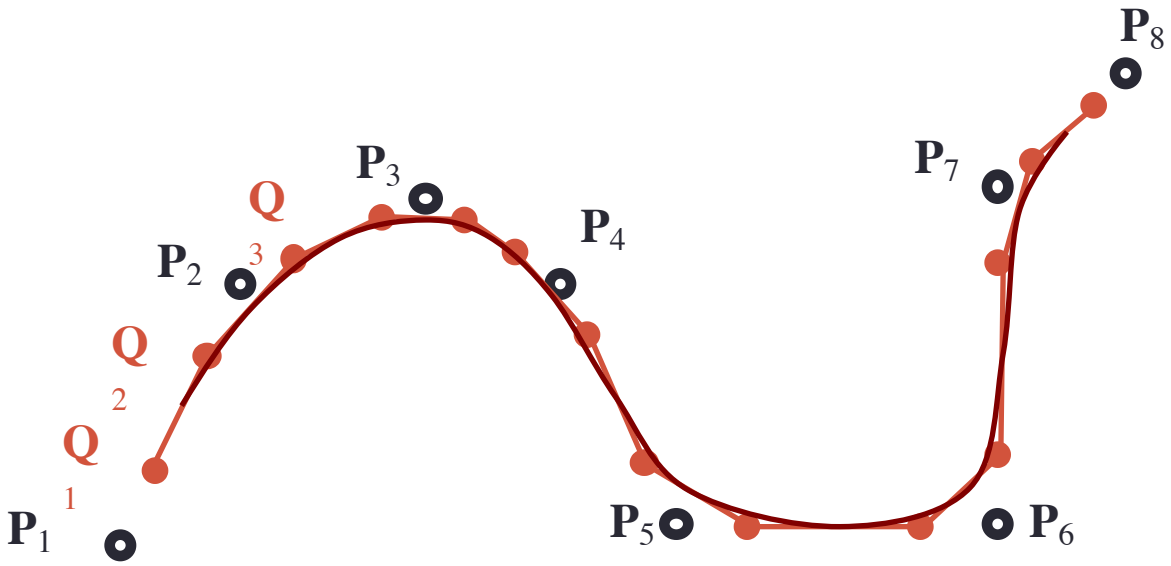

Chaikin's algorithm for Quadratic Uniform B-splines:

Given n points: $\mathbf{P}_i, i \in (1, 2, \ldots, n)$

Produce 2(n-1) points: $\mathbf{Q}_j, j \in (1, 2, \ldots, 2n-2)$

# Subdivision Curves

Uniform B-spline of order 2:




Chaikin's algorithm for Quadratic Uniform B-splines:

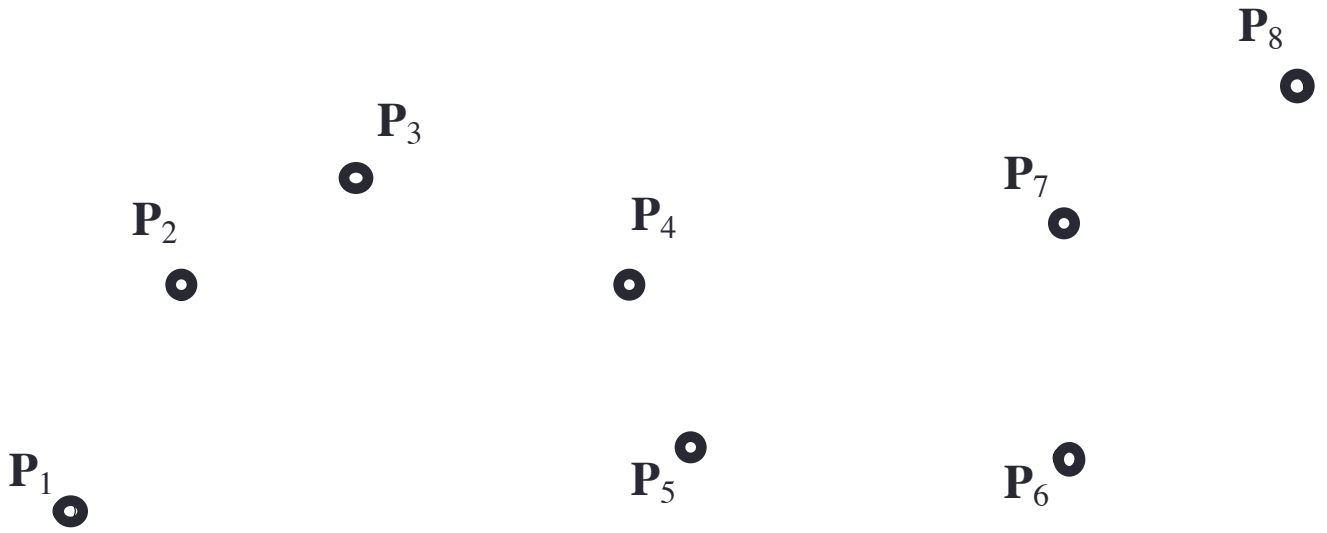
Given n points: $\mathbf{P}_i, i \in (1, 2, \ldots, n)$

Produce 2(n-1) points: $\mathbf{Q}_j, j \in (1, 2, \ldots, 2n-2)$

Let $\mathbf{P} = \mathbf{Q}$ and iterate until number of points reaches desired accuracy.

# Subdivision Curves

Uniform B-spline of order 3:

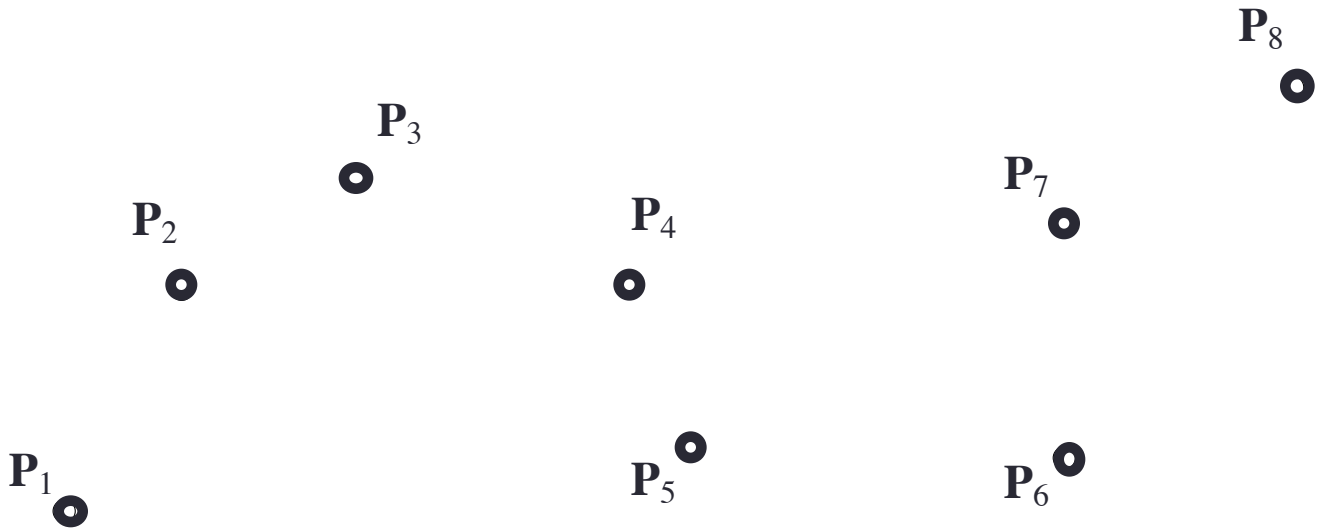



Given n points: $\mathbf{P}_i, i \in (1, 2, \ldots, n)$

Produce 2(n-1)-1 points: $\mathbf{Q}_j, j \in (1, 2, \ldots, 2n-3)$

# Subdivision Curves

Uniform B-spline of order 3:



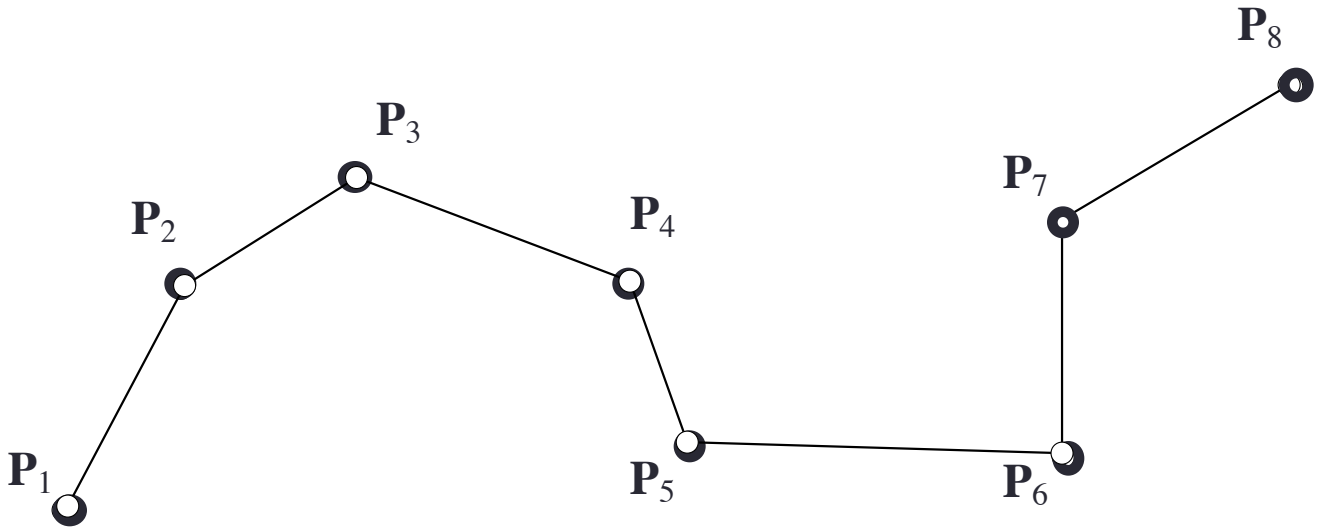

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Uniform B-spline of order 3:



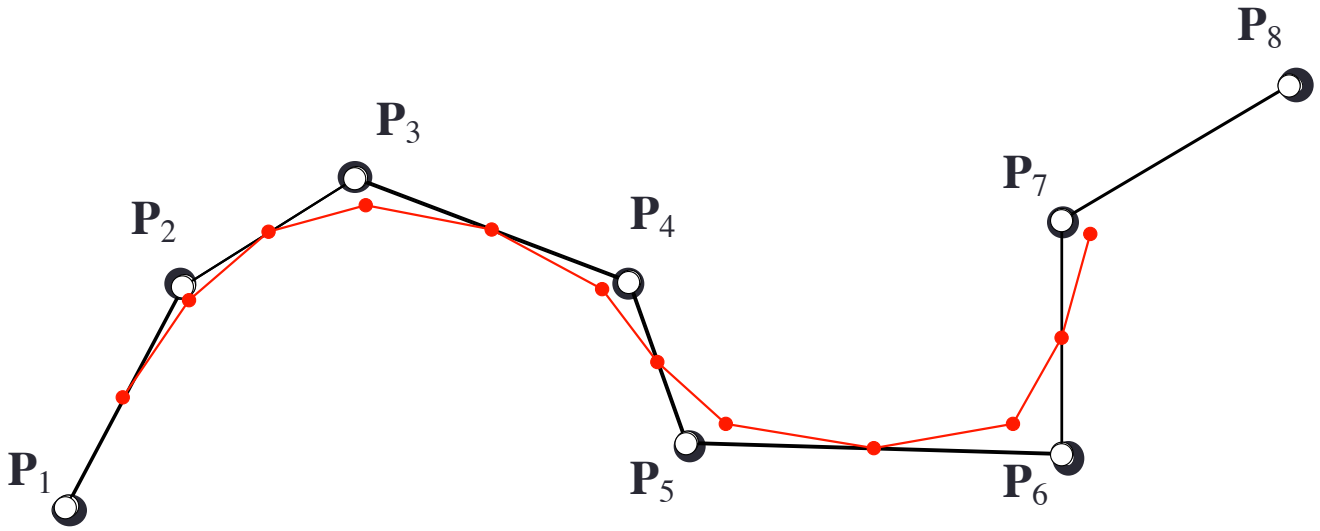

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Uniform B-spline of order 3:



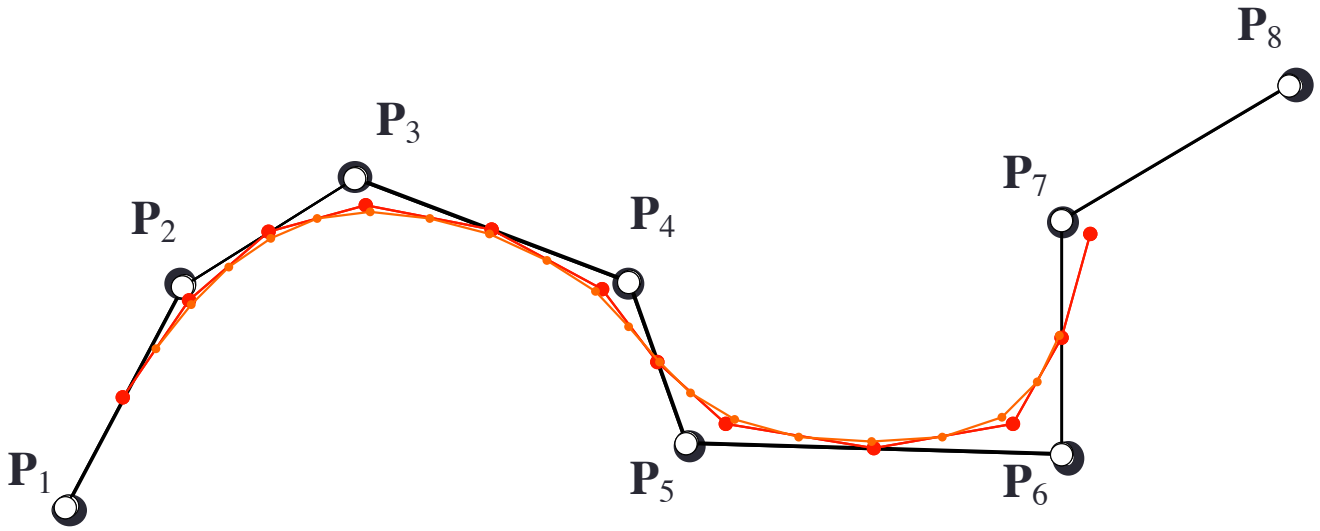

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Uniform B-spline of order 3:



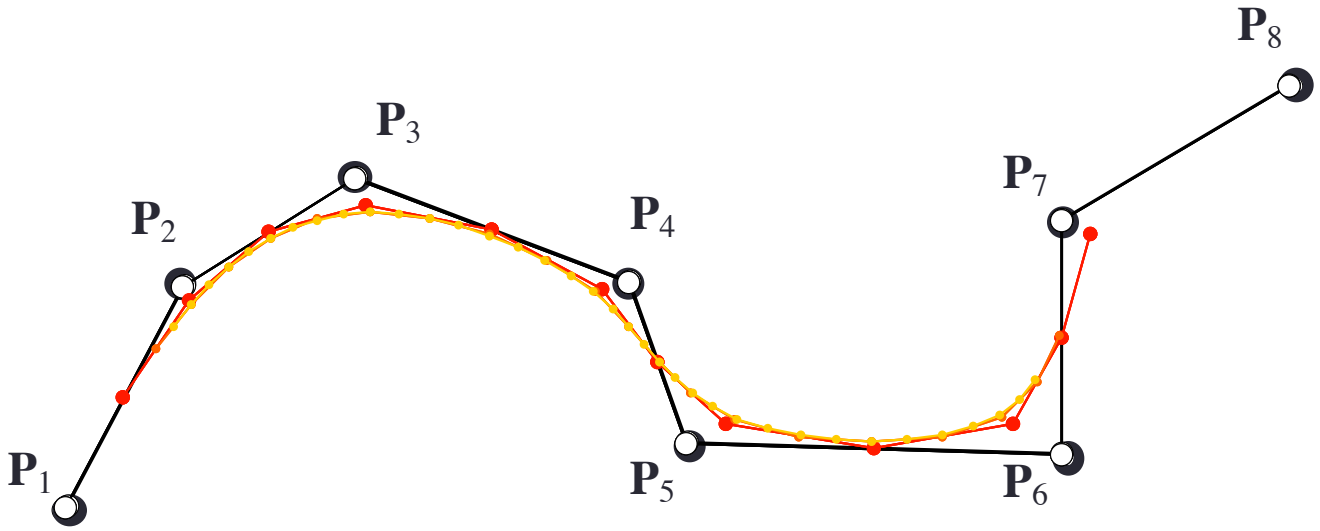

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Uniform B-spline of order 3:



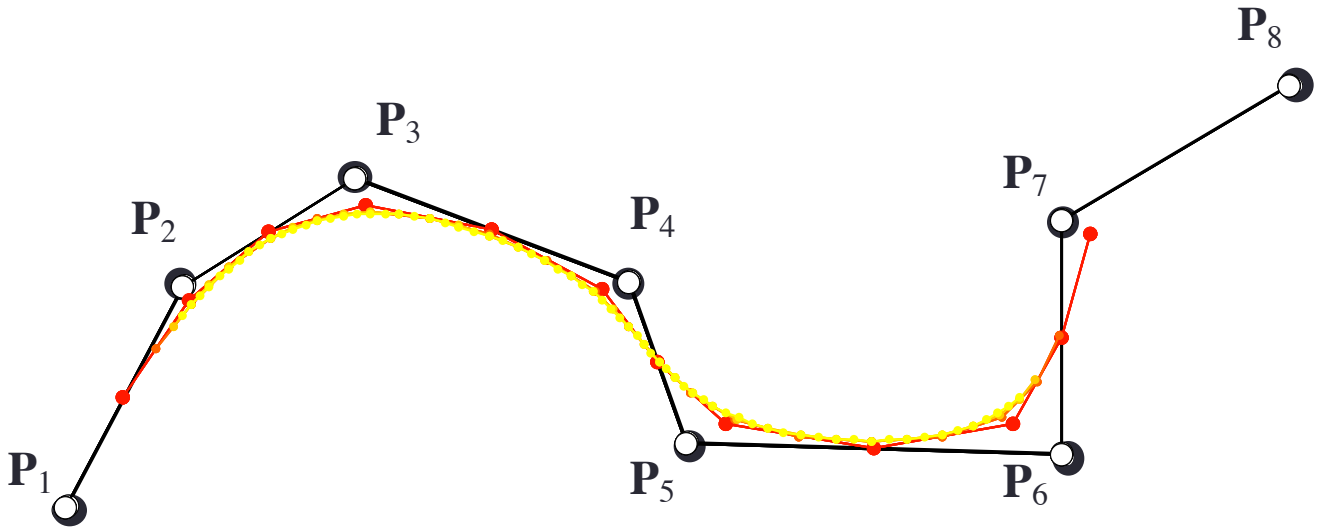

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Uniform B-spline of order 3:



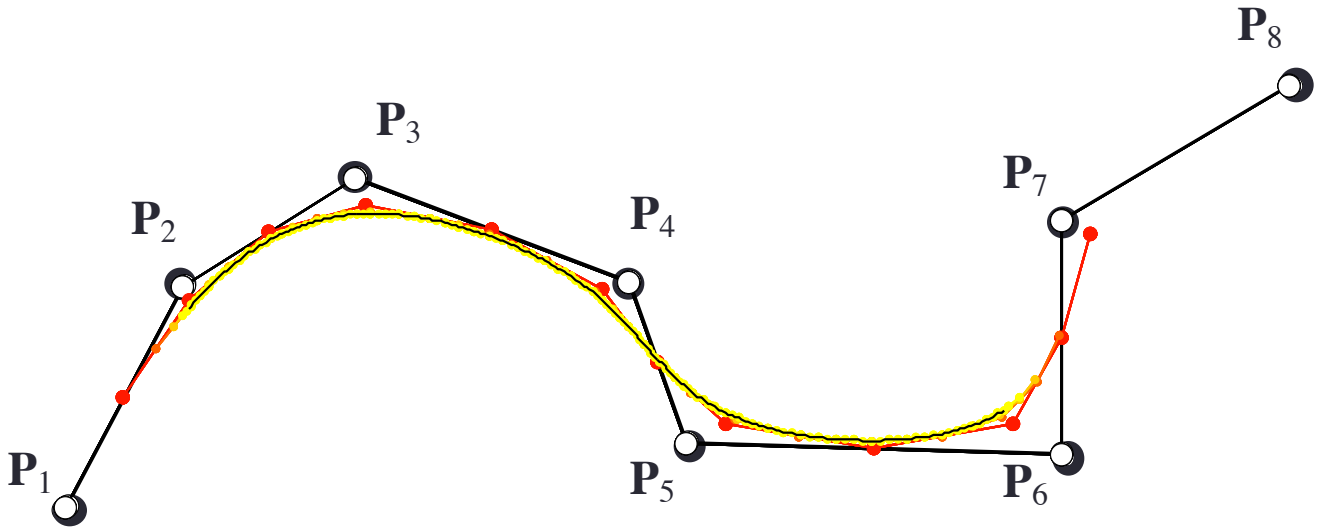

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Uniform B-spline of order 3:



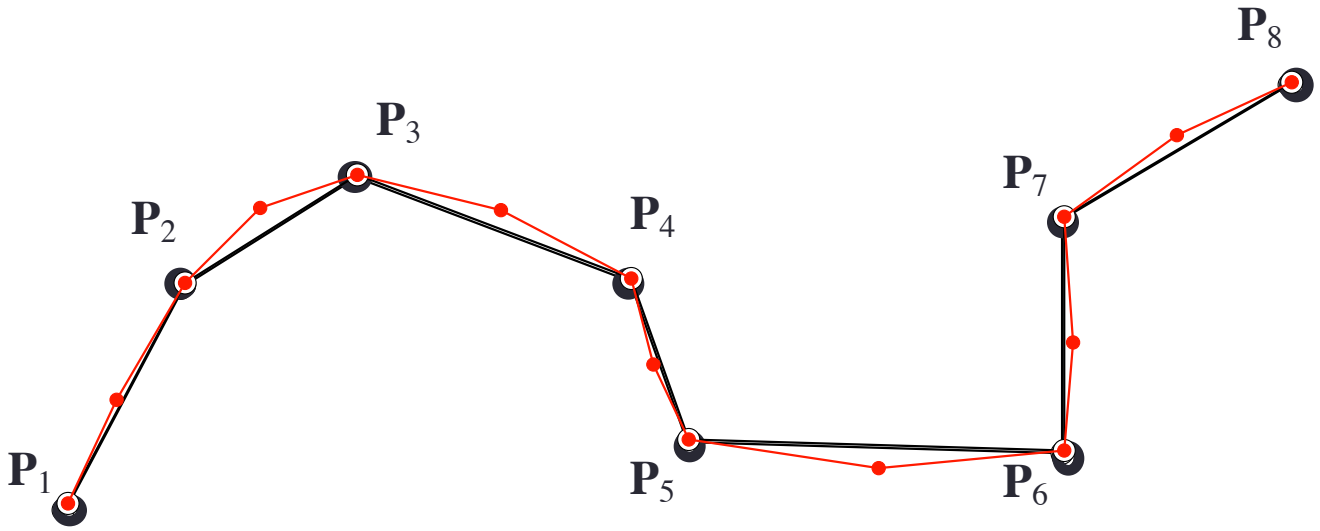

At each iteration produce 2(n-1)-1 points:

$$\mathbf{Q}_{2i-1} = \frac{1}{2}\mathbf{P}_i + \frac{1}{2}\mathbf{P}_{i+1}$$

$$\mathbf{Q}_{2i} = \frac{1}{8}\mathbf{P}_{i-1} + \frac{3}{4}\mathbf{P}_i + \frac{1}{8}\mathbf{P}_{i+1}$$

# Subdivision Curves

Interpolating curves:

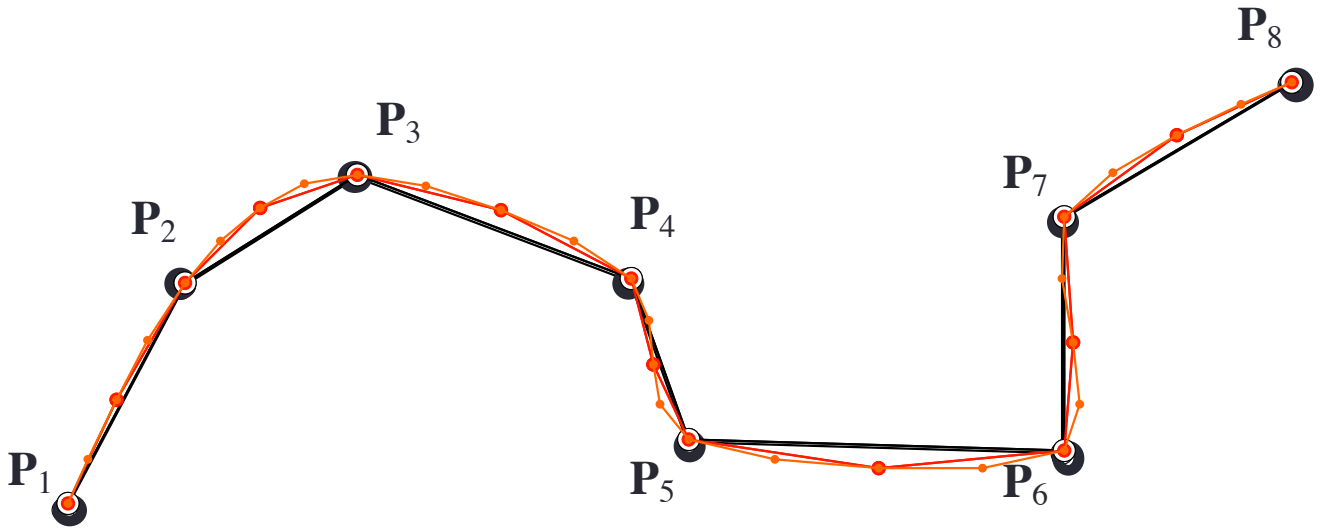



In matrix form: for every 4 consecutive old points, produce 2 new points:

$$\begin{pmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{pmatrix} = \frac{1}{16} \begin{pmatrix} 0 & 16 & 0 & 0 \\ -1 & 9 & 9 & -1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{pmatrix}$$

# Subdivision Curves

Interpolating curves:

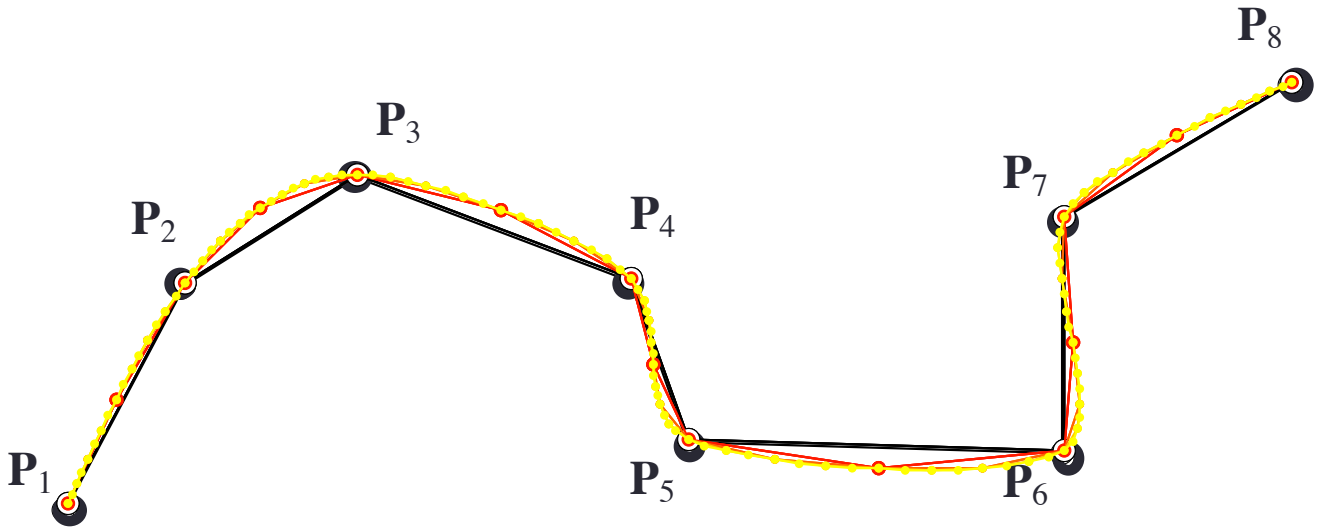



In matrix form: for every 4 consecutive old points, produce 2 new points:

$$\begin{pmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{pmatrix} = \frac{1}{16} \begin{pmatrix} 0 & 16 & 0 & 0 \\ -1 & 9 & 9 & -1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{pmatrix}$$

# Subdivision Curves

Interpolating curves:

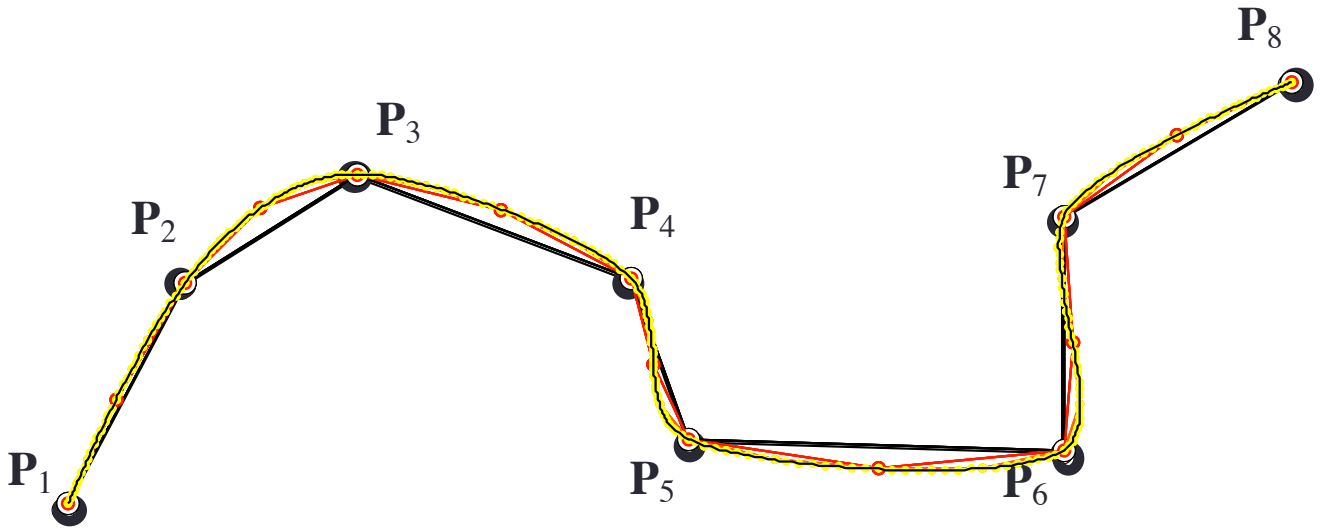



In matrix form: for every 4 consecutive old points, produce 2 new points:

$$\begin{pmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{pmatrix} = \frac{1}{16} \begin{pmatrix} 0 & 16 & 0 & 0 \\ -1 & 9 & 9 & -1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{pmatrix}$$

# Subdivision Curves

Interpolating curves:

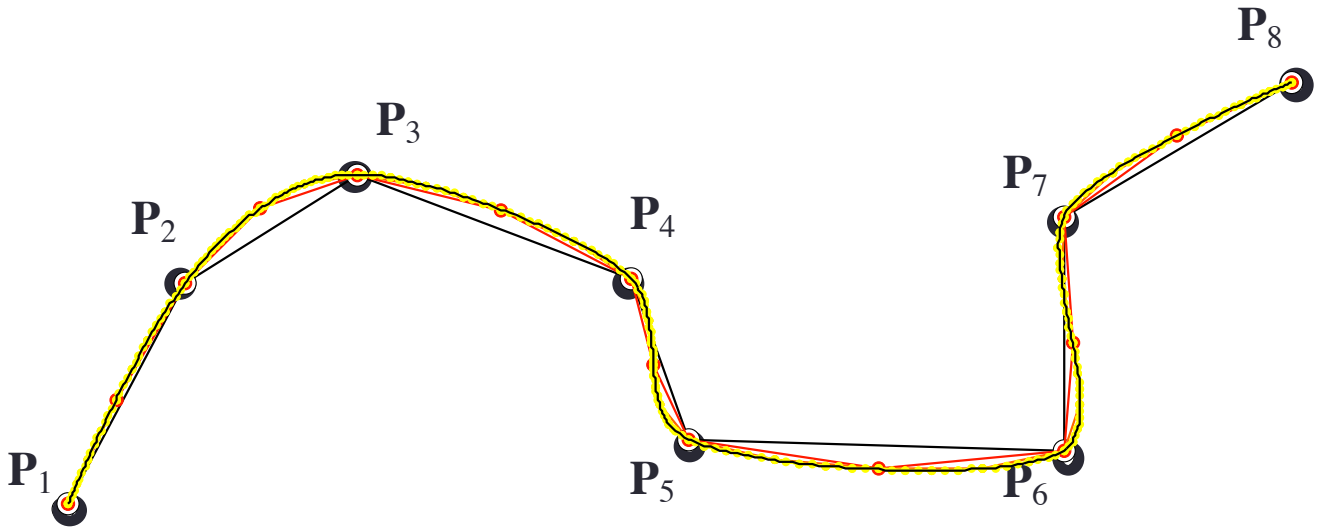



In matrix form: for every 4 consecutive old points, produce 2 new points:

$$\begin{pmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{pmatrix} = \frac{1}{16} \begin{pmatrix} 0 & 16 & 0 & 0 \\ -1 & 9 & 9 & -1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{pmatrix}$$

# Subdivision Curves

Interpolating curves:

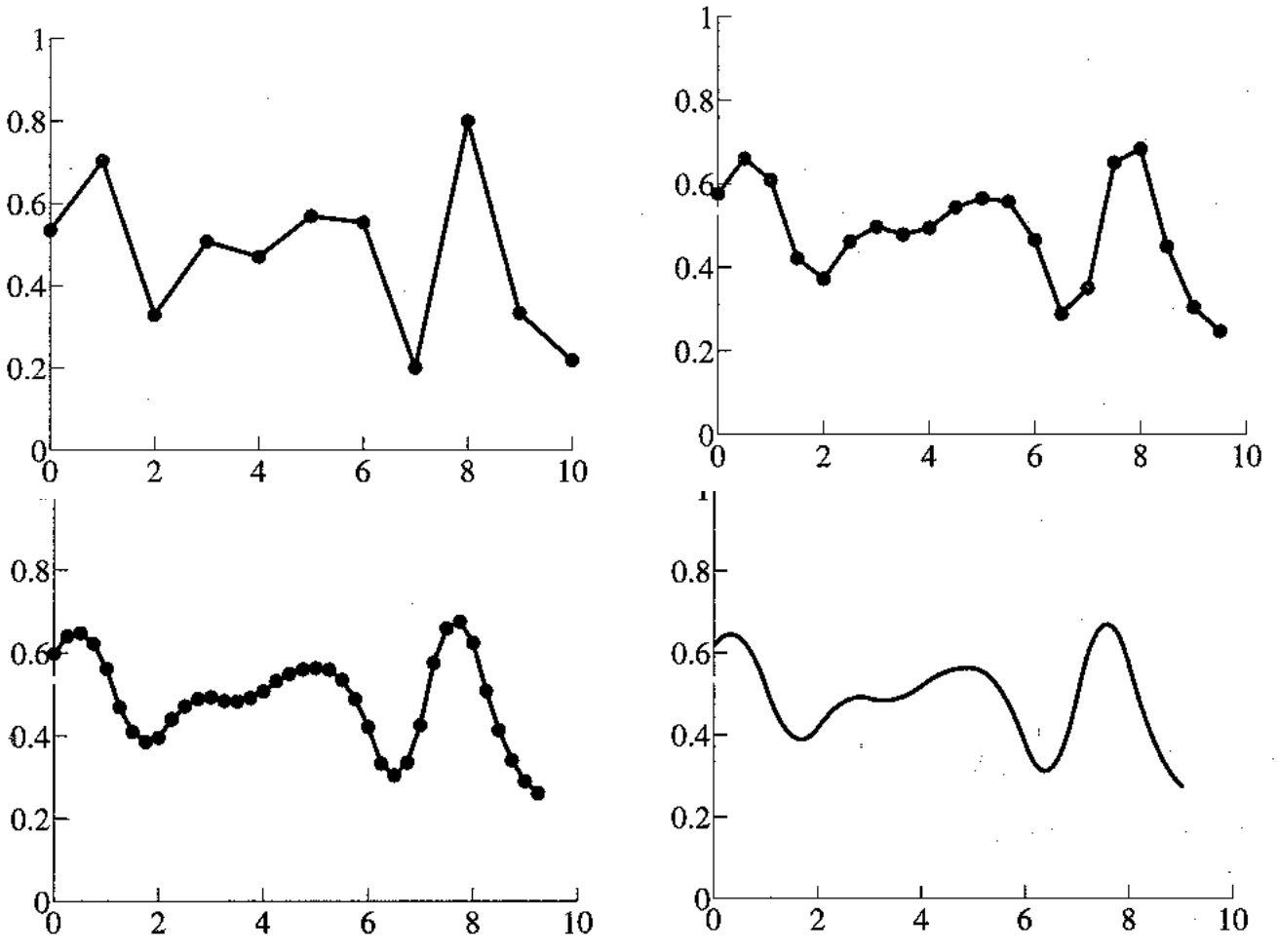



**Note:**

Before starting, make a copy of first and last points.
At each iteration, copy the first and last points.

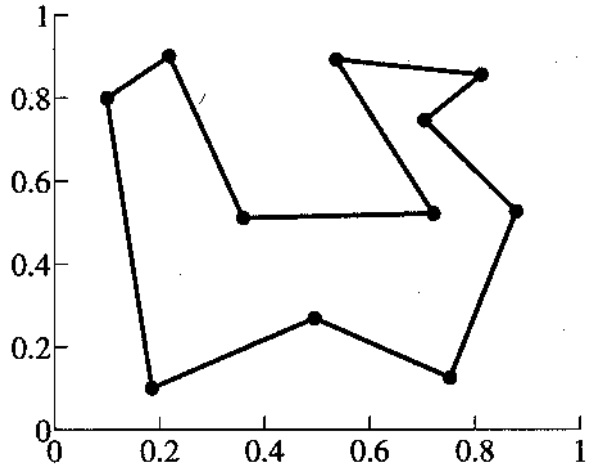
# Examples

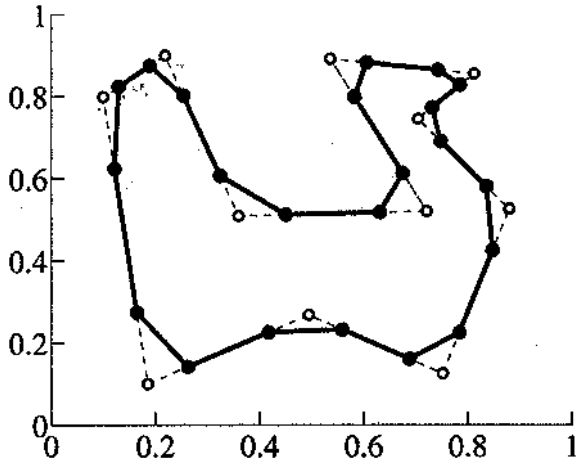
Chaikin's scheme

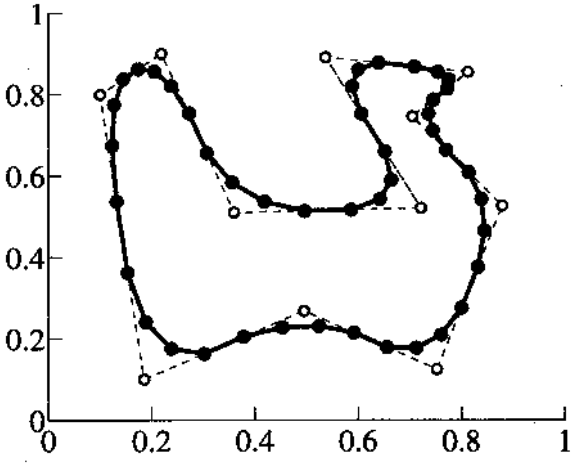


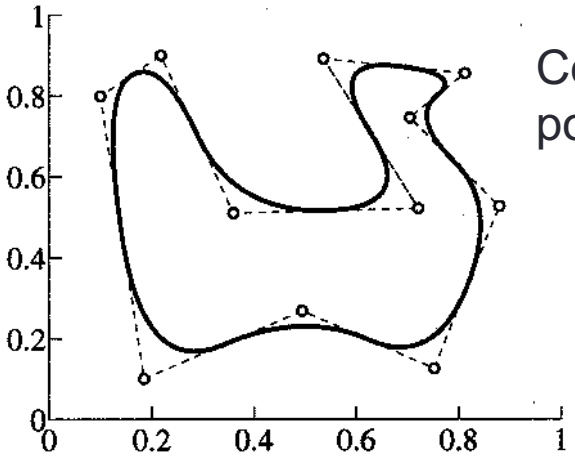
# Examples

Chaikin's scheme



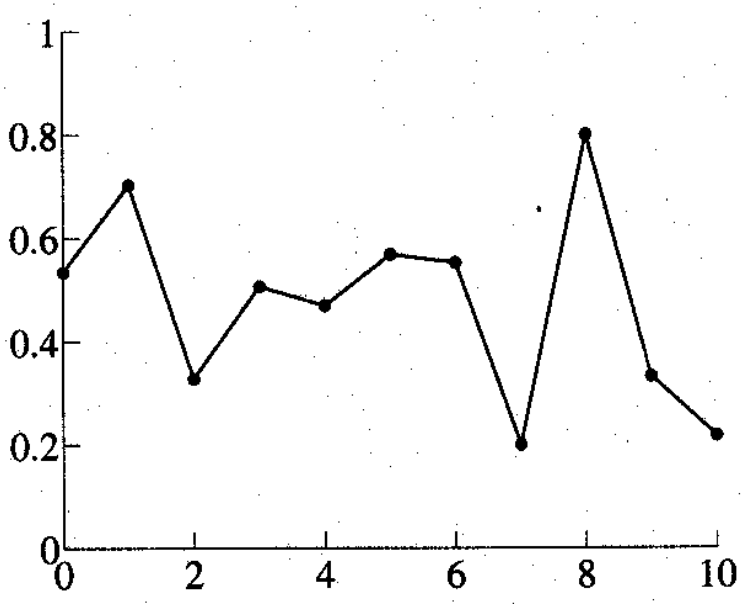



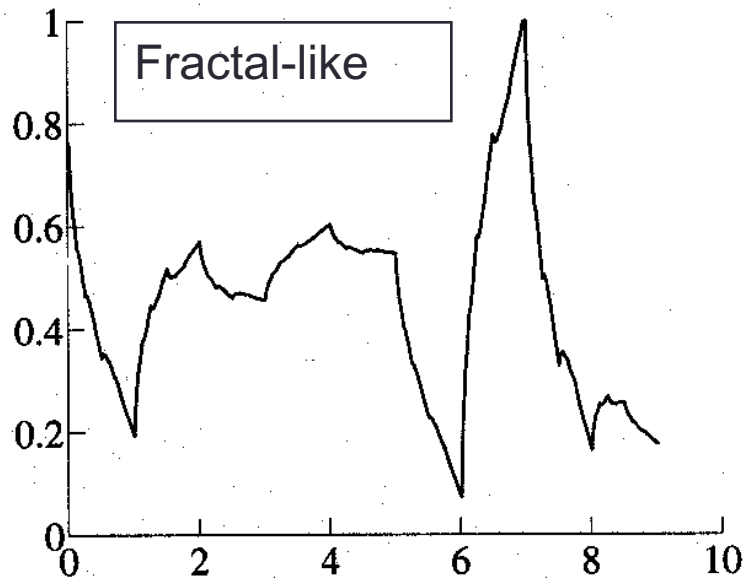

Control polygon

# Examples

Daubechies scheme

$$(r_0, r_1) = \frac{1}{2}(1 + \sqrt{3}, 1 - \sqrt{3})$$



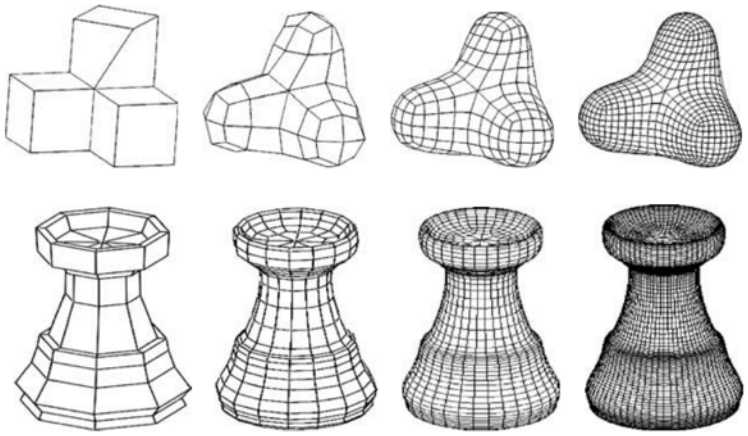



# Subdivision Surfaces

Apply the same ideas to generating smooth surfaces.

General approach:

1. Start with a control **Polytope.**
2. At each iteration refine the polytope according to some rules.
3. Stop when resolution is high enough.

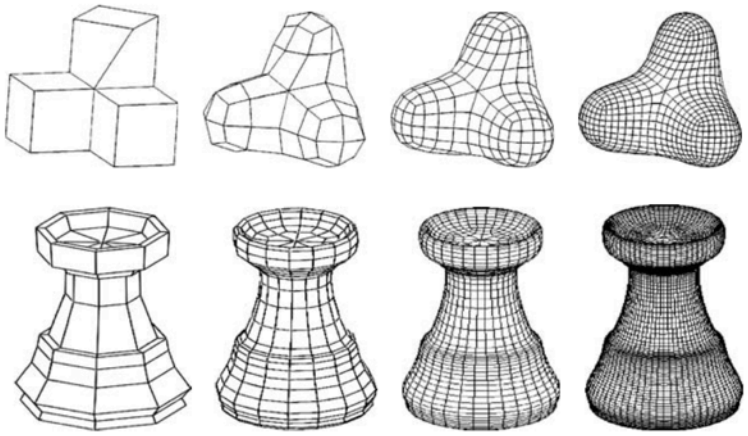


# Subdivision Surfaces

Apply the same ideas to generating smooth surfaces.

General approach:

1. Start with a control **Polytope.**
2. At each iteration refine the polytope according to some rules.
3. Stop when resolution is high enough.



# Subdivision Rules

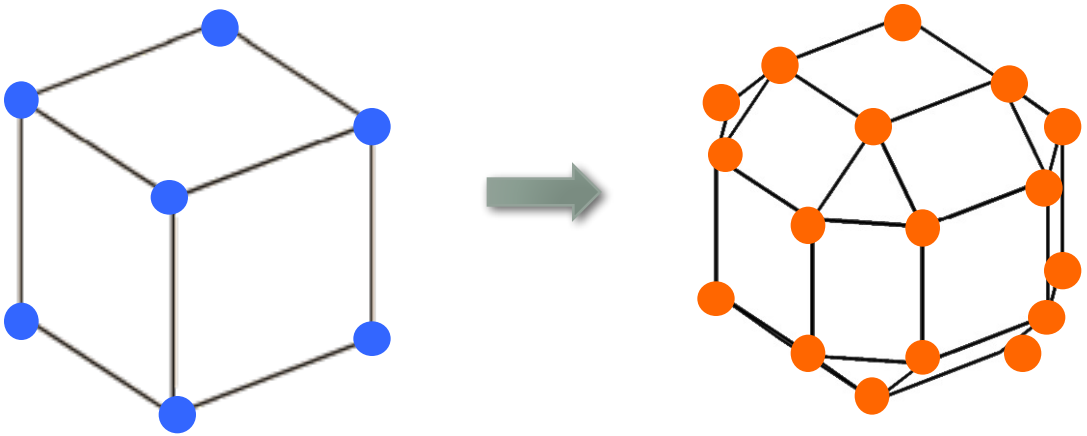
There are **topological** and **geometric** changes.

Geometric:

- How the positions of the vertices change

Topological:

- How the connectivity changes

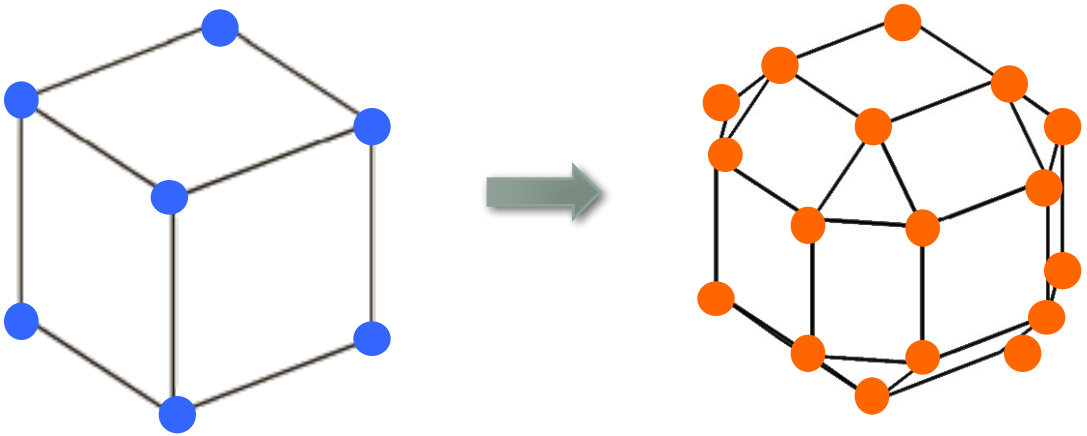


# Subdivision Rules

There are **topological** and **geometric** changes.

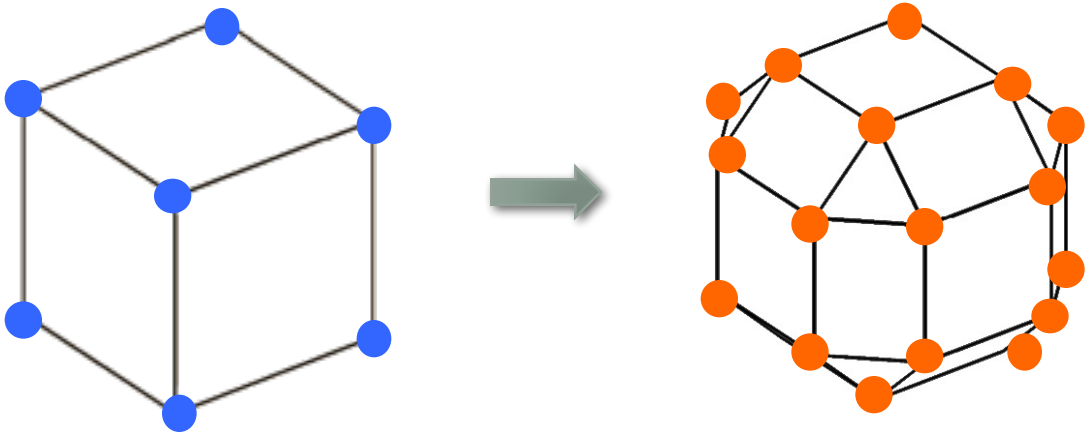
Typically, both geometric and topological changes are local:

- New vertices, edges and faces depend on a small neighborhood of old ones.



# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

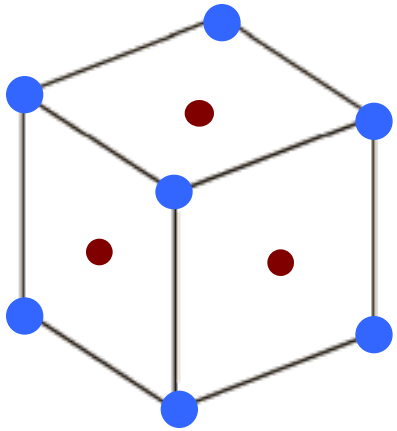


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct centroids between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

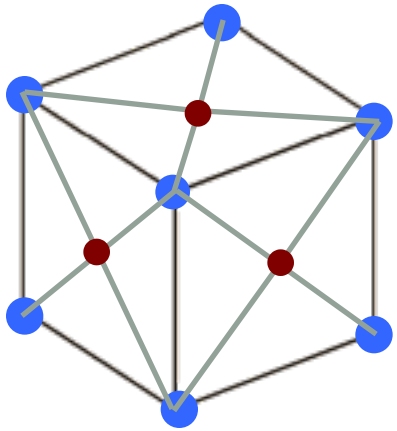


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct centroids between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

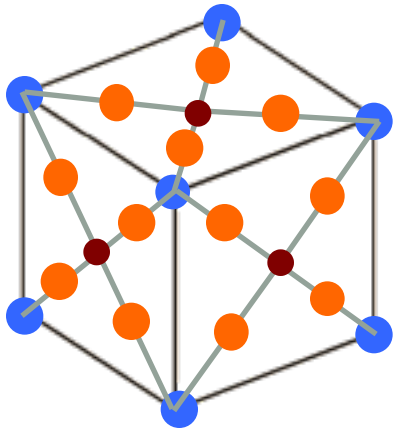


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct centroids between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

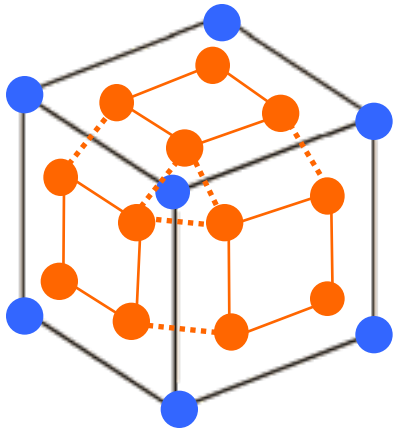


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct centroids between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

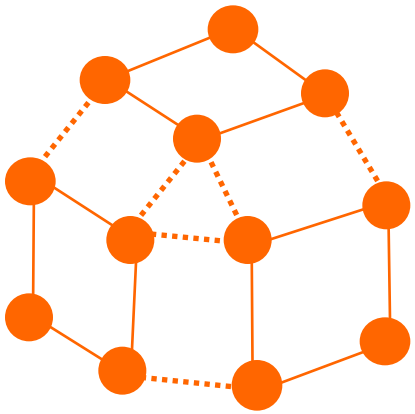


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct centroids between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

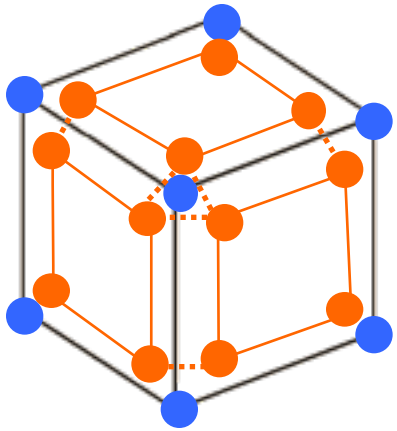


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct centroids between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

Generalization of Chaikin's corner cutting to surfaces.

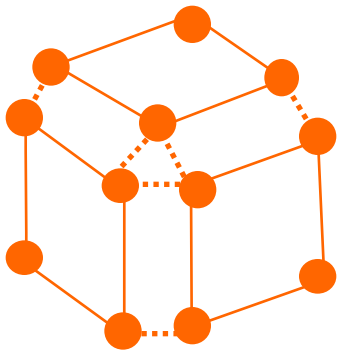


At each iteration:

1. Consider the barycenter of every (old) face
2. Construct ~~**centroids**~~ between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces

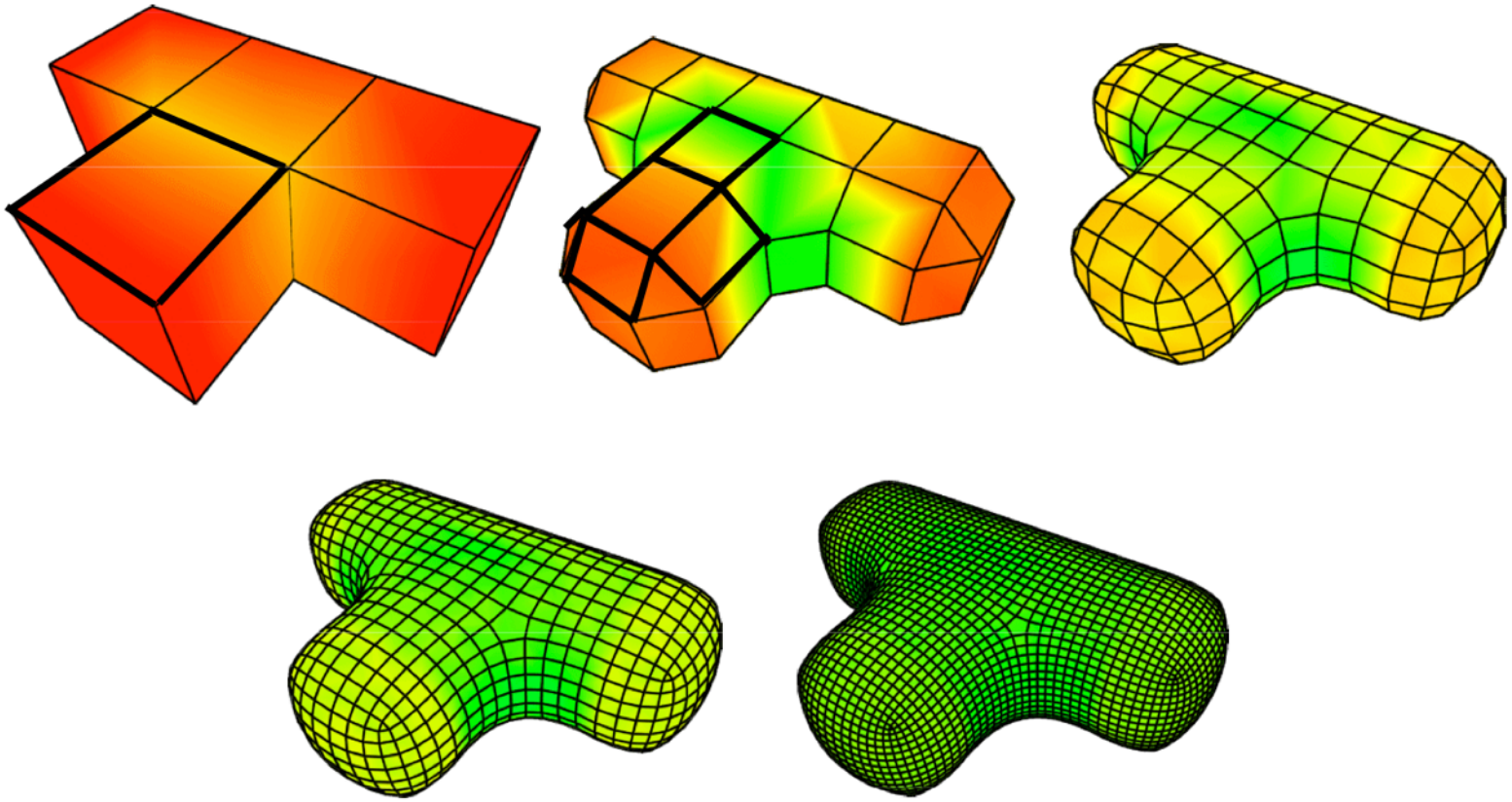
Generalization of Chaikin's corner cutting to surfaces.



At each iteration:

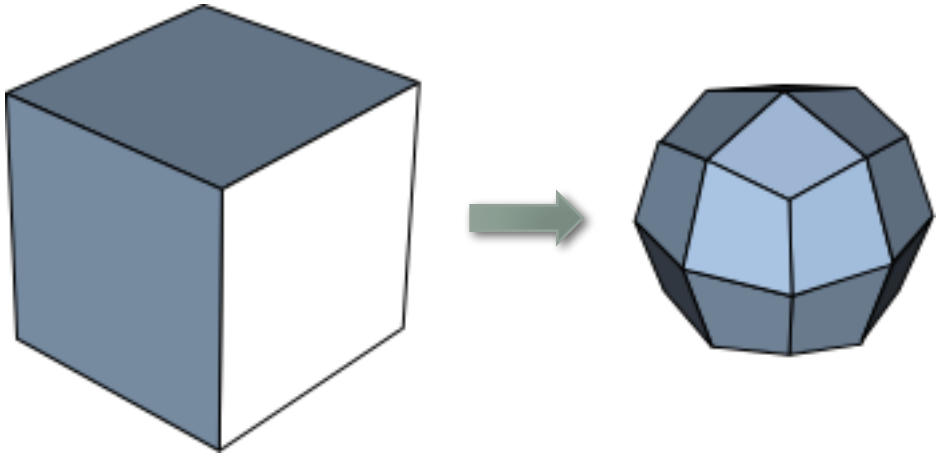
1. Consider the barycenter of every (old) face
2. Construct ~~**centroids**~~ between the center and old vertices.
3. Connect them in a natural way.
4. Restart.

# Doo-Sabin subdivision surfaces



# Catmull-Clark subdivision surfaces

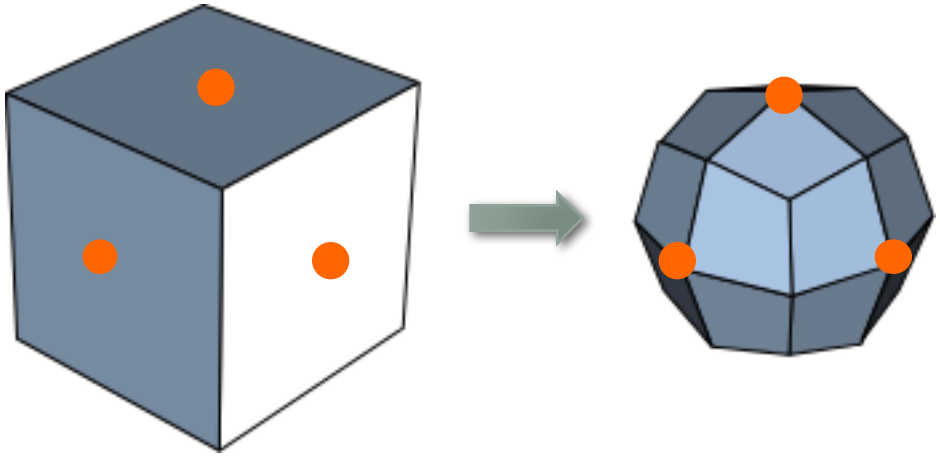
Generalization of cubic spline subdivision to surfaces.



- Approximating Scheme
- Small support stencil (just immediate neighbors)
- Limit surface is 2nd-order continuous except at extraordinary vertices
- Subdivision scheme used in all modern Pixar films

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.

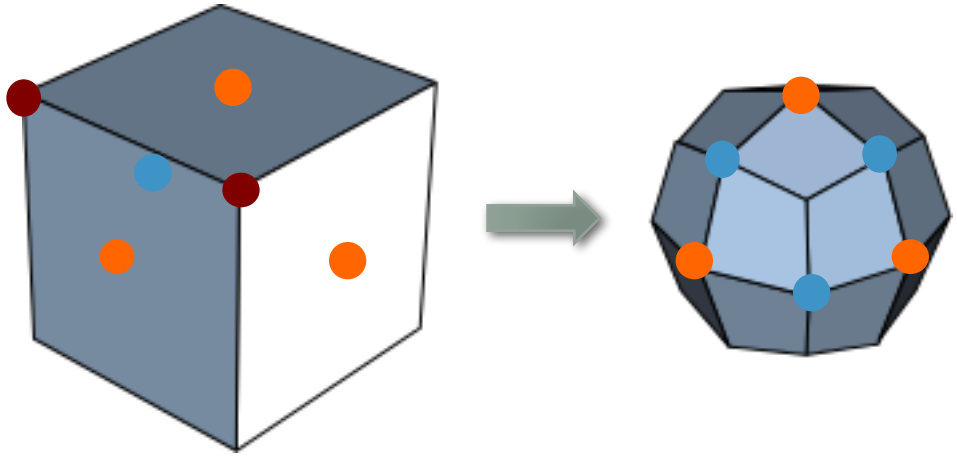


At each iteration:

1. Construct Face vertices: barycenters of old faces.
2. Construct Edge vertices.
3. Update existing vertices.
4. Connect them in a natural way.
4. Restart.

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.

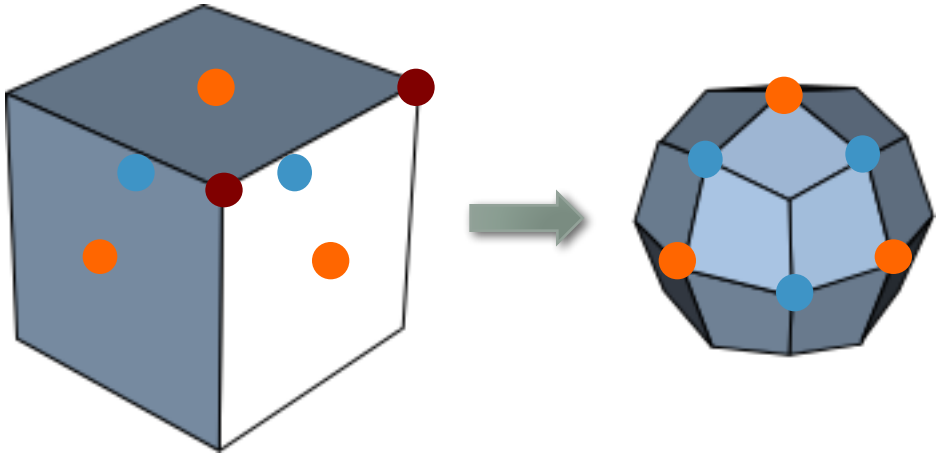


At each iteration:

1. Construct Face vertices: barycenters of old faces.
2. Construct Edge vertices: average of the old edge vertices and the associated face vertices

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.

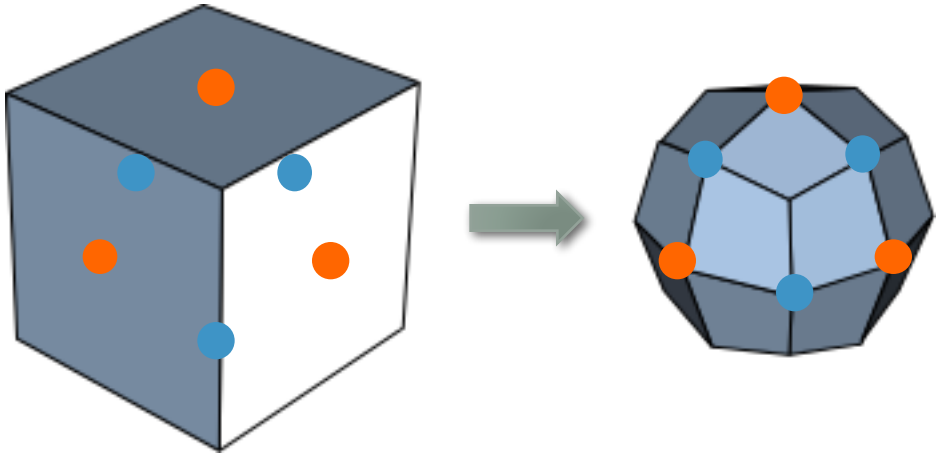


At each iteration:

1. Construct Face vertices: barycenters of old faces.
2. Construct Edge vertices: average of the old edge vertices and the associated face vertices

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.

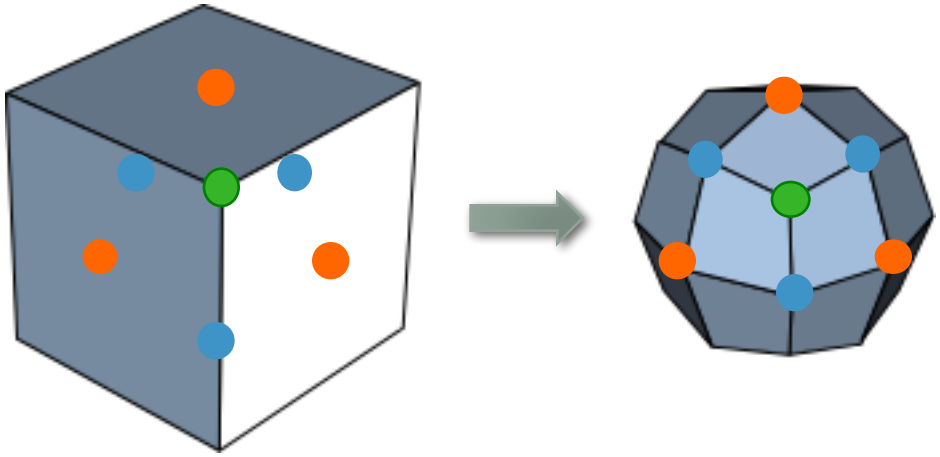


At each iteration:

1. Construct Face vertices: barycenters of old faces.
2. Construct Edge vertices: average of the old edge vertices and the associated face vertices

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.

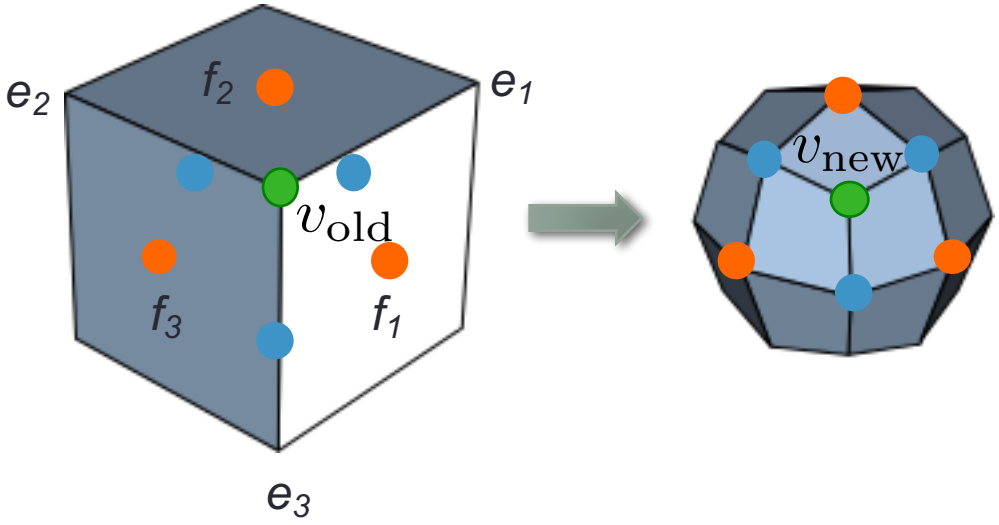


At each iteration:

1. Construct Face vertices: barycenters of old faces.
2. Construct Edge vertices.
3. Update existing vertices.
4. Connect them in a natural way.
4. Restart.

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.




At each iteration:

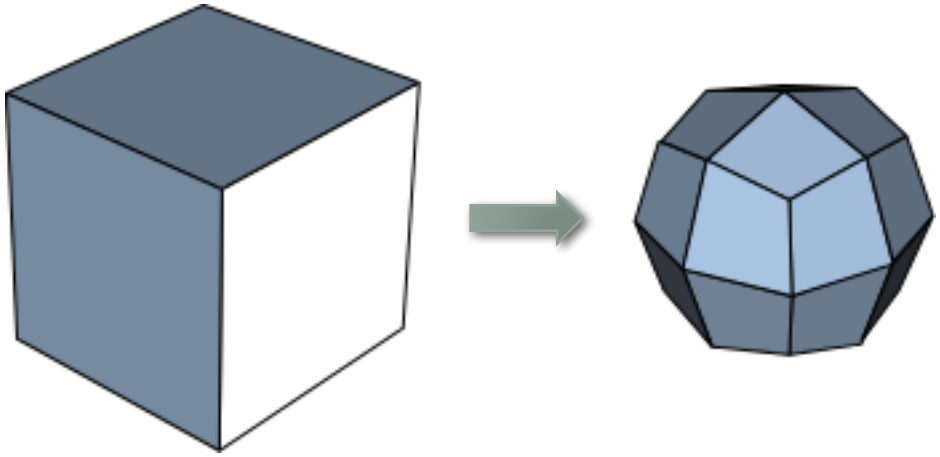
1. Construct Face vertices: barycenters of old faces.
2. Construct Edge vertices.
3. Update existing vertices.

$$v_{\text{new}} = v_{\text{old}} + \frac{1}{n^2}\sum_{j=1}^{n}(e_j - v_{\text{old}}) + \frac{1}{n^2}\sum_{j=1}^{n}(f_j - v_{\text{old}})$$

$e_j$: **old** vertex incident along edge j
$f_i$: **new** (orange) vertex on face j
$n$: number of incident edges.

# Catmull-Clark subdivision surfaces

Generalization of cubic spline subdivision to surfaces.

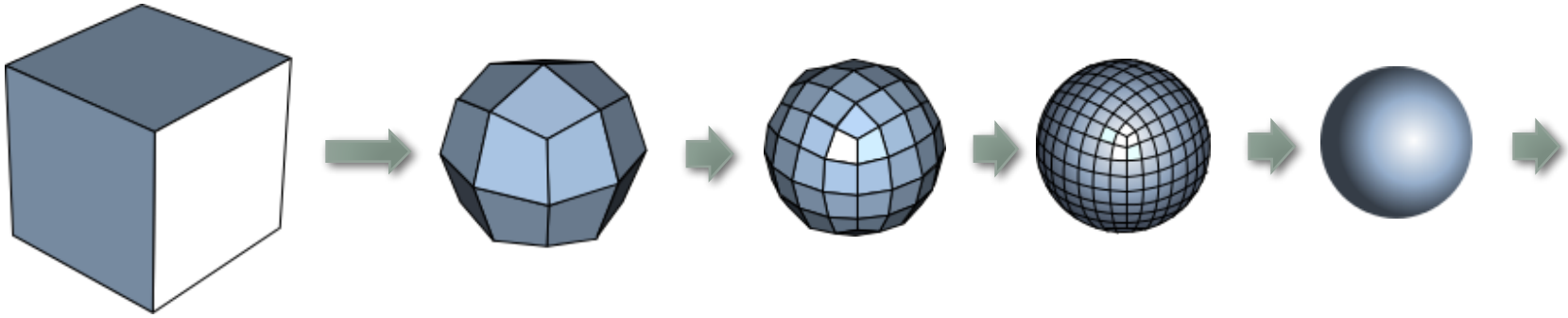


At each iteration:

1. Construct Face vertices.
2. Construct Edge vertices.
3. Update existing vertices.
4. Connect them in a natural way.
4. Restart.

# Catmull-Clark subdivision surfaces

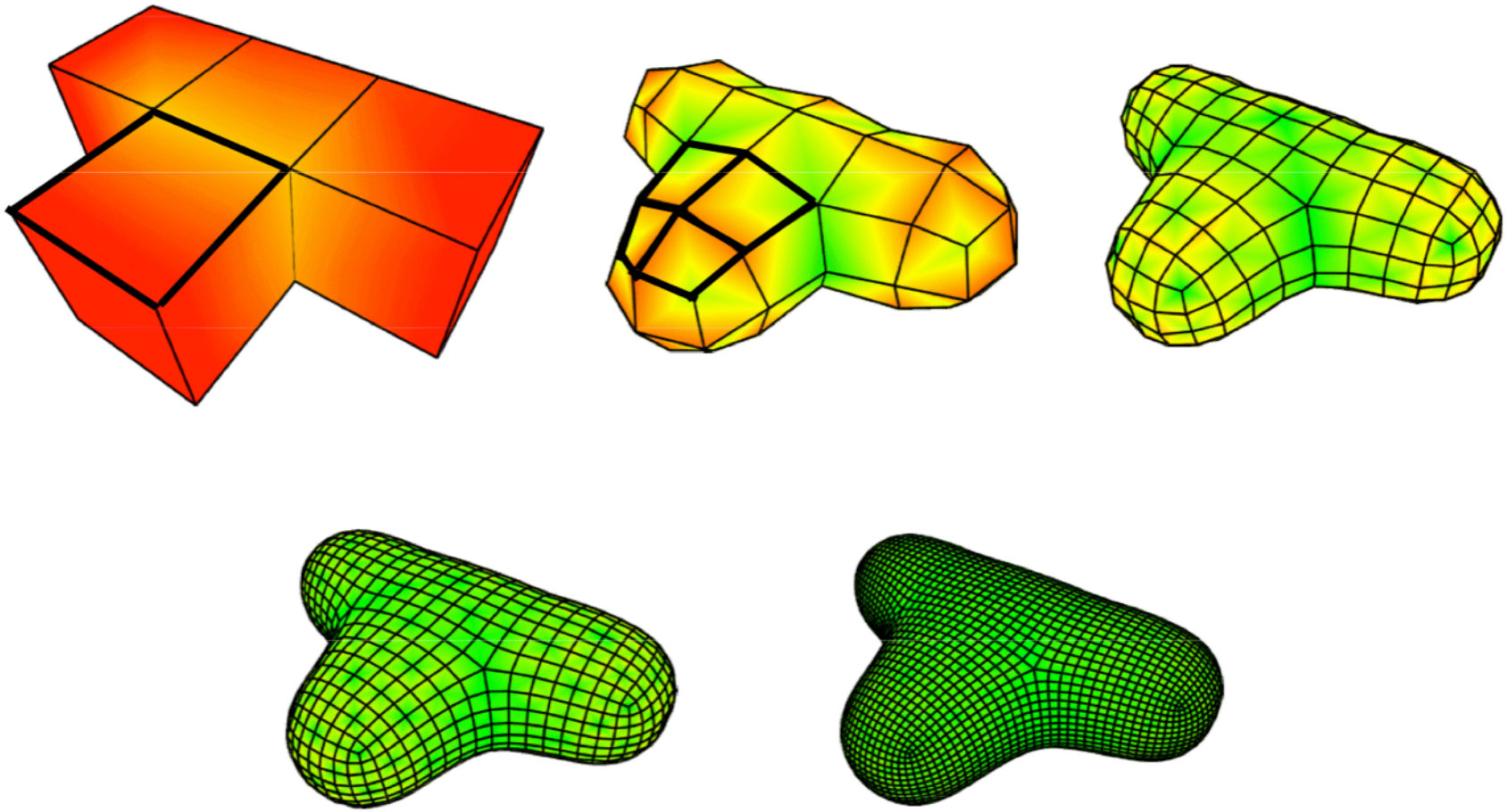
Generalization of cubic spline subdivision to surfaces.



At each iteration:

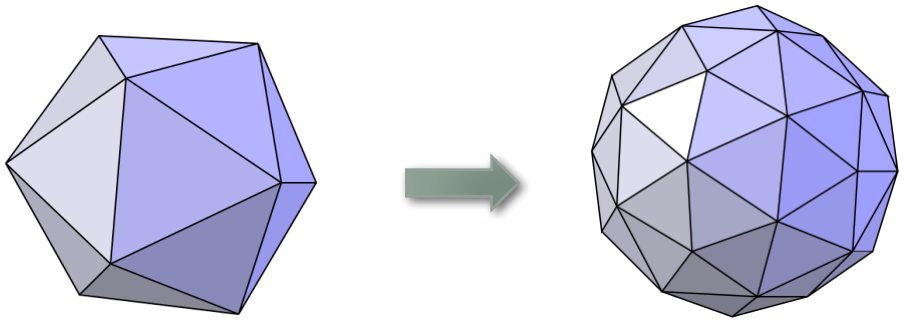
1. Construct Face vertices.
2. Construct Edge vertices.
3. Update existing vertices.
4. Connect them in a natural way.
4. Restart.

# Catmull-Clark subdivision surfaces



# Loop subdivision surfaces

Triangle-based subdivision:

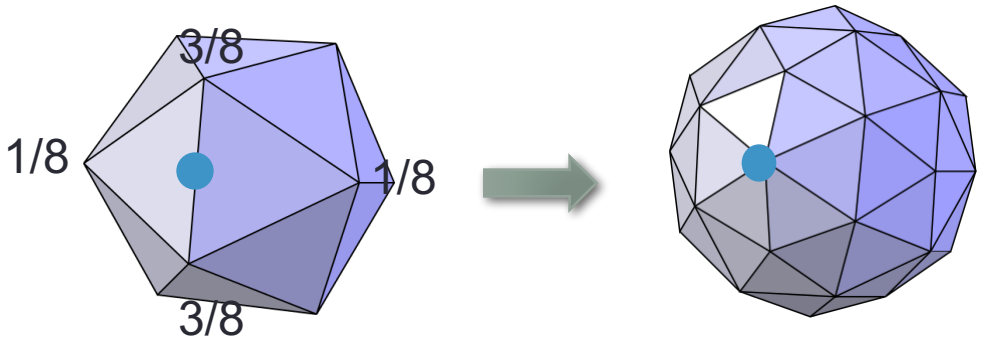


At each iteration:

1. Construct Edge vertices.
2. Update existing vertices.
3. Connect them in a natural way.
4. Restart.

# Loop subdivision surfaces

Triangle-based subdivision:



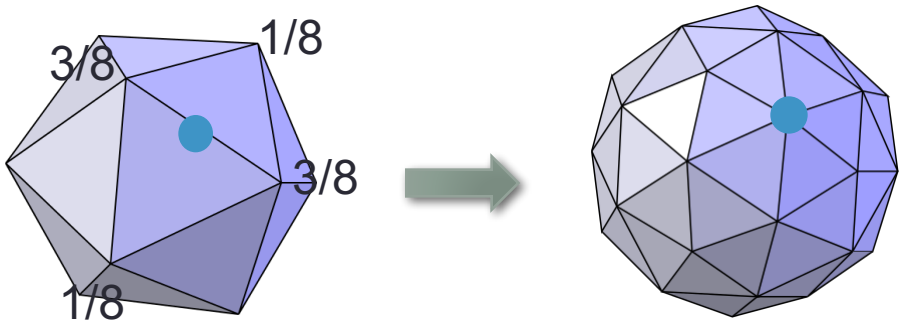

At each iteration:

1. Construct Edge vertices.

$$e_i = \frac{3}{8}v_{e1} + \frac{3}{8}v_{e2} + \frac{1}{8}v_{t1} + \frac{1}{8}v_{t2}$$

# Loop subdivision surfaces

Triangle-based subdivision:



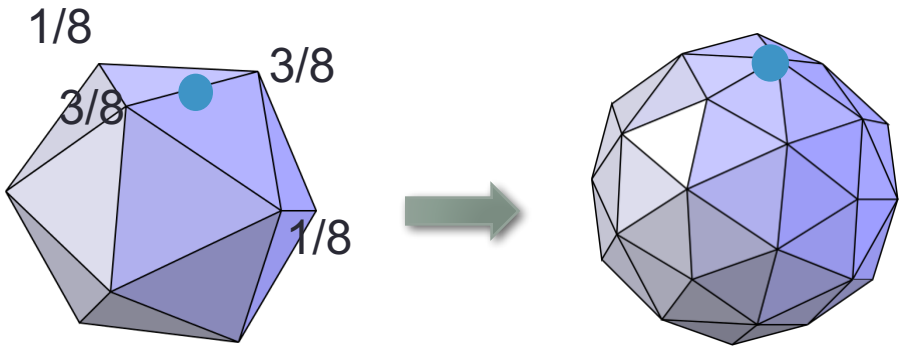

At each iteration:

1. Construct Edge vertices.

$$e_i = \frac{3}{8}v_{e1} + \frac{3}{8}v_{e2} + \frac{1}{8}v_{t1} + \frac{1}{8}v_{t2}$$

# Loop subdivision surfaces

Triangle-based subdivision:



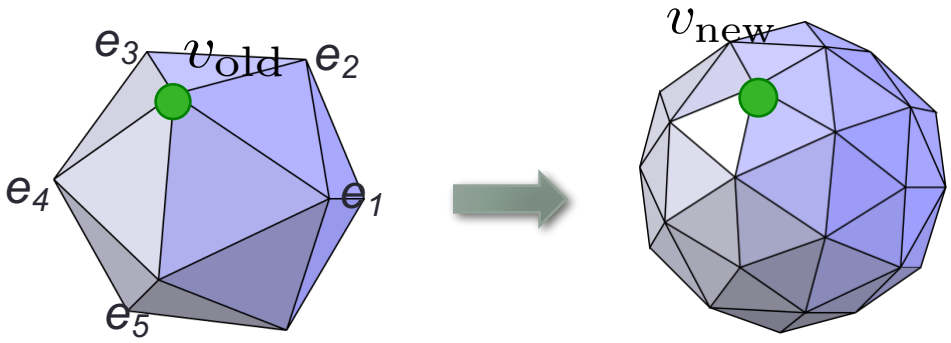

At each iteration:

1. Construct Edge vertices.

$$e_i = \frac{3}{8}v_{e1} + \frac{3}{8}v_{e2} + \frac{1}{8}v_{t1} + \frac{1}{8}v_{t2}$$

# Loop subdivision surfaces

Triangle-based subdivision:




At each iteration:

1. Construct Edge vertices.
2. Update existing vertices:

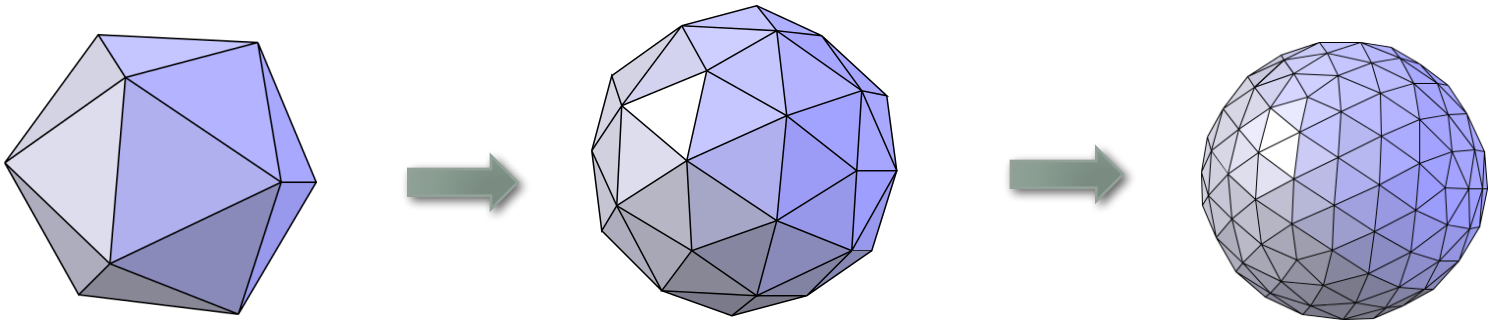
$$v_{\text{new}} = (1 - \alpha n) v_{\text{old}} + \alpha \sum_{j=1}^{n} e_j$$

$$\alpha = \begin{cases} \frac{3}{16} & \text{if } n = 3 \\ \frac{3}{8n} & \text{if } n > 3 \end{cases}$$

$e_j$: old vertex incident along edge j
$n$: number of incident edges.

# Loop subdivision surfaces

Triangle-based subdivision:

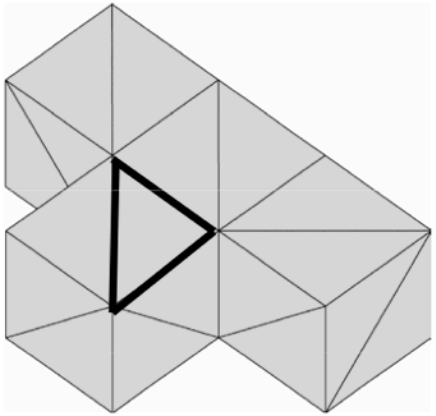


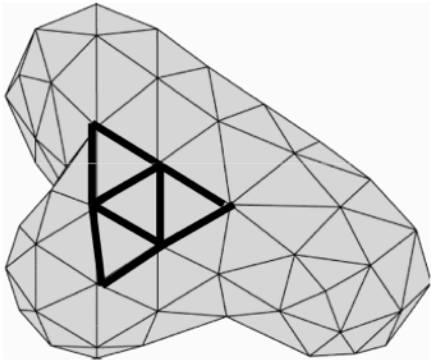
At each iteration:

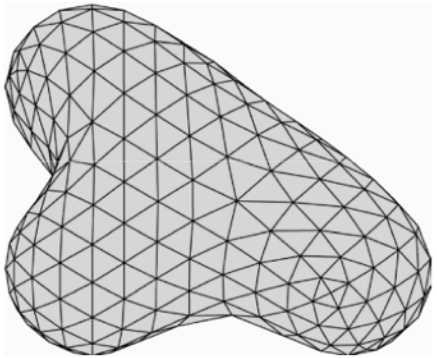
1. Construct Edge vertices.
2. Update existing vertices.
3. Connect them in a natural way.
4. Restart.

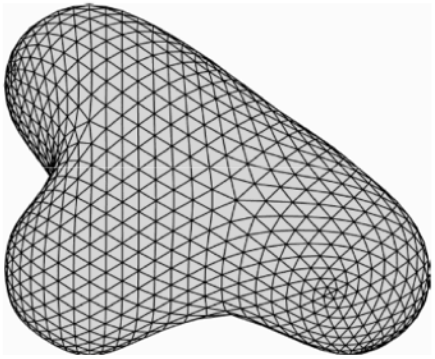
Attention: different update rules on the boundary.

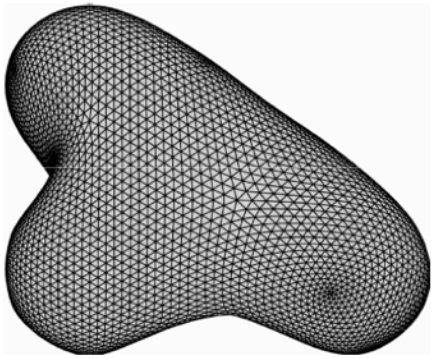
# Loop subdivision surfaces







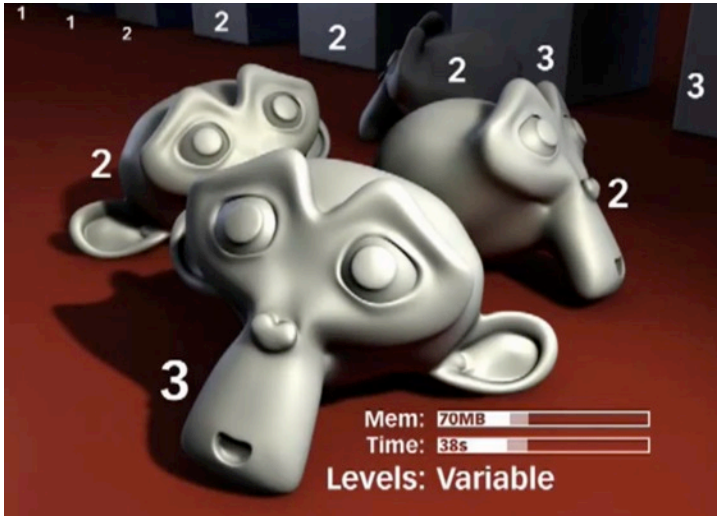




# Conclusions

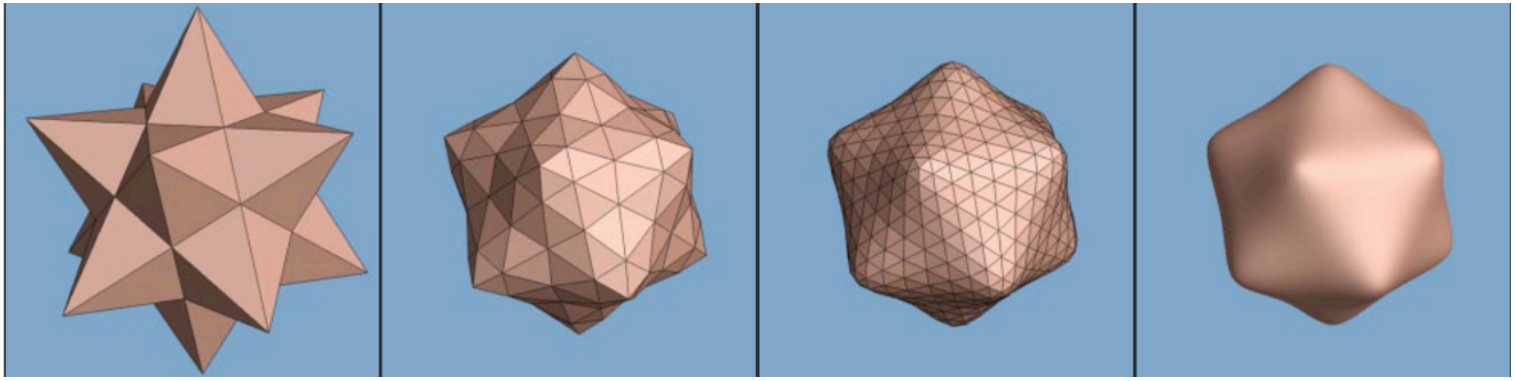
Subdivision surfaces:

- Allow simpler modeling
  - Major strength: surfaces of arbitrary topology
  - Limit surfaces are smooth
  - Control mesh is typically simple and intuitive
- Adapt to user's needs
  - Render only at required level-of-detail
- Usability
  - Compact representation
  - Simple and efficient code

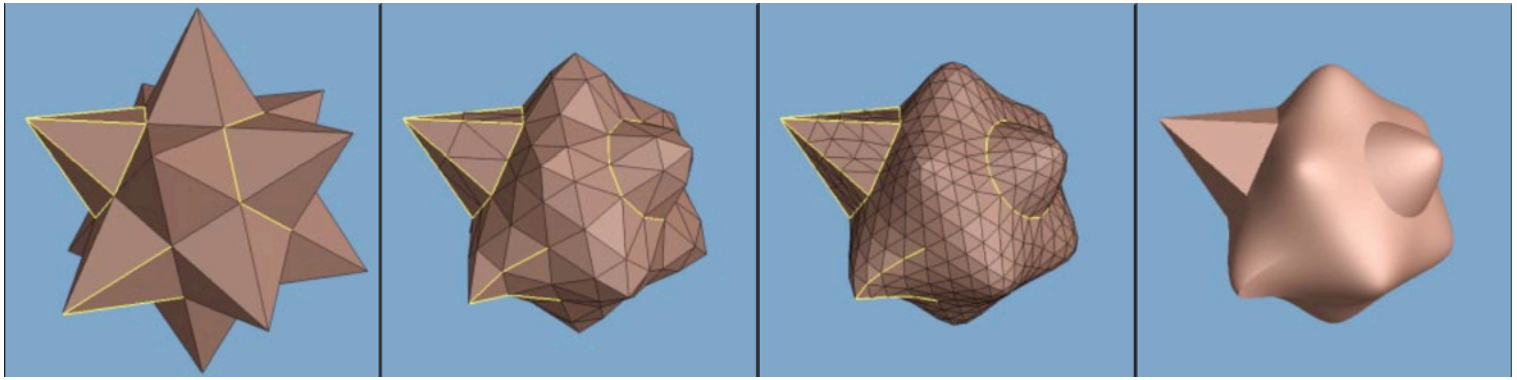


# Extensions

Piecewise-smooth subdivision surfaces:



Allow some sharp edges to remain



Hoppe et al. *Piecewise Smooth Surface Reconstruction*, SIGGRAPH '94