Chapter 4 Foundations And Representations

报告人：刘源

Introduction

- point-based surface representations
- mathematical and algorithmic fundamentals

Contents

4.1 Surface Reconstruction

- 4.1.1 Overview
- 4.1.2 Normal Estimation
- 4.1.3 Implicit surface methods
- 4.1.4 Voronoi methods
- 4.1.5 surface evolution methods

4.1 Overview

$$\text{Object} \rightarrow \text{point-cloud} \rightarrow \text{surfaces} \begin{cases} \textit{triangle mesh} \\ \textit{pathes} \\ \textit{zero-set} \\ \ldots \end{cases}$$

for Point-cloud dense enough : many different approaches

Choice : *depends partly on* **input** and **output**

noisy input point cloud with many outliers

- *filter the outliers*
- *extract as much information as possible from the noisy data*

point cloud produced by a series of modeling operations

- *noise free*
- *methods that interpolates the input points could be used*

Point clouds captured using commercial laser range scanners : fall somewhere in between

- *few outliers*
- *some noise : scanner artifacts at sharp edges and alignment error*

Choice : output

watertight surface : a surface bounding a closed solid

- *necessary for finite element analysis, rapid prototyping, parameterization, · · ·*
- *watertight constraint often makes for easier, or more robust, algorithms*

surfaces with boundaries

- *a scan of a human face · · ·*

Compared by theoretical analysis

The framework :

- *assume that the input is densely enough*
- *prove the result recovers the correct topology and approximate geometry*

4.1.2 Normal Estimation

- reconstruction is easier with reliable normal directions

Methods

Hoppe et al

1. find the k-nearest neighbors of point p : $N_k(p)$
2. fine the *total least squares best-fitting plane* H to $N_k(p)$
3. take the normal of H as the normal of p

drawback

need uniform distribution, lead to trouble when points are in slices

An alternative : use all points within distance r of p, but lead to the same difficulty.

Mitra et al.

Choosing r adaptively at different points of P, since **no single best choice of r for a given input P**

- areas of high curvature, $r \nearrow$, error$\nearrow$
- areas of high noise , $r \nearrow$, error$\searrow$

Thus, r is chosen according to the local curvature and noise level.

Using the Voronoi diagram

The Voronoi cells of a point p on the exterior of P are elongated in the direction perpendicular to the surface, the vector from p to its **pole** is a good estimate of the normal.

4.1.3 Implicit surface methods

produce a function f :

$$f() : \begin{cases} > 0, & \text{outside the object} \\ < 0, & \text{inside the object} \end{cases} \Longrightarrow$$

S is the zero level-set of $f() : \{f = 0\}$.

- output : always the watertight boundary of a solid
- should be computed on a domain large enough to surround the input point set P and S

An alternative : take the domain to be a thin shell surrounding P, result in a surface with boundary.

Voxel-based Methods

the implicit surface method was implemented on a voxel grid

Hoppe et al.

1 estimate the normal at each p by fitting $N_k(p)$

2 orient the normal by traverse a spanning tree of P

3 $f(x) = dist(x, \text{tagent plane of } p \text{ nearest to x})$

4 extract piecewise-linear surfaces from f=0

drawback : output may suffer from holes

Curless and Levoy

laser range data input : a grid of points in $x - y$ plane with a depth value z.

1 per patch : connecting points adjacent in the $x - y$ plane

2 each patch : associate with a with a directional distance function

3 form f() : blending the distance functions, with normalized Gaussian weights

Efficiency : limiting the domain to a thin shell, return surfaces with boundary.

Noise processing : confidence assigned to points, lower value in sharp features.

Basis Functions

Implicit surfaces $\rightarrow$ a weighted sum of basis functions, commonly radical basis :

$$\hat{f}(x) = \sum_i c_i \theta(\| x - p_i \|)$$

c_i : solve to interpolate or approximate the constraints

Advantage : output surface is smooth and attractive

Notice :

The function $f \equiv 0$ always satisfy the constraint, thus constraints **inside** *or* **outside** *the surface are necessary.*

If normals available, we can place additional off-surface points offset from it in both normal directions.

Indicator Function

Another choice for f:

$$f(x) = \begin{cases} 1 & x \text{ is inside the object} \\ 0 & x \text{ is outside the object} \end{cases}$$

Thus, gradient of f:

$$\nabla f = \begin{cases} 0 & x \notin S \\ \vec{n} & x \in S \end{cases}$$

the problem becomes a Poisson problem : compute f from its gradient field.

MLS and MPU with Local Functions

Idea :

- construct many "little" functions f_i locally
- then blending the to form f

weight function $\omega_i : dist(p_i, x) \nearrow, \omega_i \searrow$

$$f(x) = \frac{\sum_i \omega_i(\| x - p_i \|) f_i(x)}{\sum_i \omega_i(\| x - p_i \|)}$$

This function minimize

$$\sum_i \frac{\omega_i(\| x - p_i \|)}{\sum_i \omega_i(\| x - p_i \|)} (f(x) - f_i(x))^2$$

which is just the notion of **moving least square approximation**.

Notice :*if* $\omega_i = 0$ *when* **far** *from* x, f *can be compute efficiently.*

In application, ω_i is often chosen to be Gaussian, and f_i **sufficiently far** *from* x *is discarded.*

MPU(multilevel partition of unity) : *one implementation of this idea*

- octree is used to partition the space
- low-degree polynomials approximations is constructed on each leaf

functions associated with interior nodes is used for lower-resolution approximation of the space.

4.1.4 Voronoi methods

a particular strength : provide proofs of correctness

Definition 1 (ϵ-sample) *A set of sample points on the surface is an ϵ-sample if, for every point x on the surface, there is a sample within distance $\epsilon f(x)$ of x.*

The Voronoi cells of an ϵ-sample : long, thin, and perpendicular to the surface.

two poles : the two "ends" of these long thin cells, lie near the medial axis

Observation :

- *vector "p to pole" is close to normal*

thus used to approximate the surface normals

Power Curst Method : uses the weighted Voronoi diagram of the poles

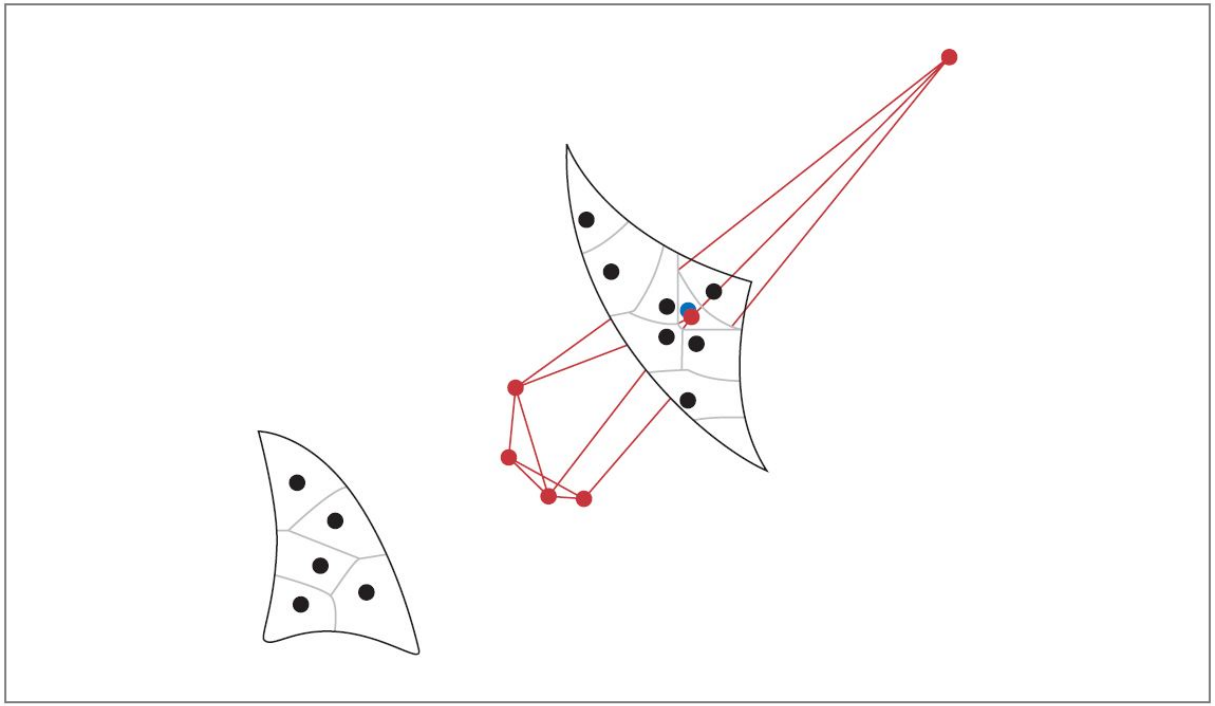


Figure 4.2: The 3D Voronoi diagram of points sampled from a smooth 2D surface. The intersection of the 3D cells with the surface is shown in black, and the edges of one Voronoi cell, belonging to the blue point, are shown in red. Notice that the Voronoi cell is long and skinny, with its long axis perpendicular to the surface. The ends of the Voronoi cell are located near the medial axis.

Advantages

- *do not require normals : poles provide a good approximation*
- *find normals with consistent orientation*

disadvantages

- *computation is expensive in time and space*
- *most of the triangulation is thrown away*

compute only the necessary part ?

ball pivoting algorithm

rolls a ball of fixed radius around the outside of the point cloud

when the ball rest on three input samples, connect them with a triangle

- not so robust as the entire triangulation

4.1.5 Surface evolution methods

Idea :

gradually deform a simple input surface

using rules to maintain its structure

attract it to the input data

Chen and Medioni : *an early straightforward implementation*

(need a explicit representation, say, a triangle mesh)

1 initialize : a small ball inside the point cloud

2 expand with ”balloon forces” until it reaches the input points

3 when a sample ”reaches” the surface, anchor it

the forces:

- inflation force in the normal direction
- spring forces between neighboring vertices

limitation : object should be homeomorphic to a sphere

Level-set method *(Osher and Sethian)*

- evolving surface is represented by a level-set in an implicit 3D function
- describe the evolution by a PDE
- solve the evolution numerically

For example, the evolving surface Γ is obtained in minimizing the surface quality functional :

$$E(\Gamma) = \left[\int_{x \in \Gamma} d^m(x, P)\, ds \right]^{1/m}$$

Notice that the functional is reduced by bringing the surface closer to the sample set P.

Level-set method can usually handle the topological changes, but may get stuck in local minima in some situation. (the following figure). A good **initial estimate** generate better results.

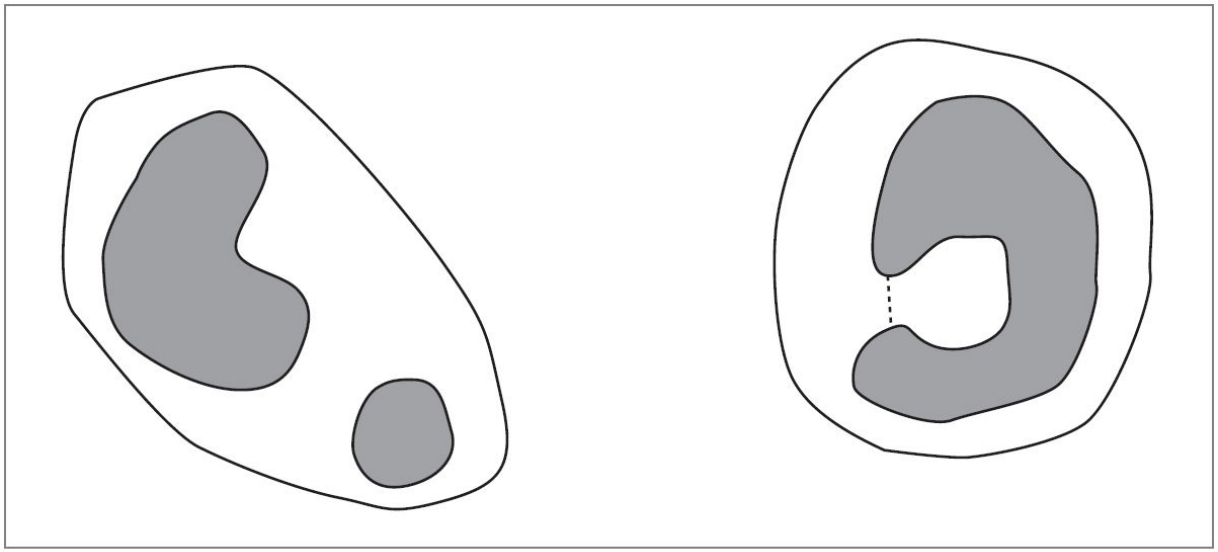


Figure 4.3: Surface evolution starting from a bounding surface. Most topological changes are handled gracefully, but sometimes it can get stuck in an undesired local minimum. (*Left*) The outer surface is attracted to the object boundary. At some moment, it covers both the connected components, and the connecting channel shrinks to have zero volume, and then disappears, leaving a correct surface representation. (*Right*) The shrinking outer surface will get stuck in a local minimum including the dotted line and not the interior of the cavity; moving the dotted line into the cavity increases the surface area and requires more energy.

4.1.6 Conclusion

- *different methods applies to different requirements*
- *taking more time to do a better job is appropriate trade off : the noisy input*
- *memory efficiency is probably more important for really large inputs*
- *an improved understanding of the point distribution will be helpful*

4.2 Moving Least Squares-Based Surface Representations

- 4.2.1 Overview
- 4.2.2 Notation and terms
- 4.2.3 Interpolation and approximation of functional data
- 4.2.4 Normals
- 4.2.5 Implicit surfaces from points and offset points
- 4.2.6 Implicit surface from points and tangent frames

4.2.1 Overview

Global structures

$\surd$ lead to very good reconstruction results

$\times$ always consider all of the data, even for **local operations** : inefficient in time and space

Only local algorithms have the premise to be efficient when used to perform certain local operations on very large point sets.

Here we consider the so-called ***moving least squares(MLS) approach.***

4.2.2 Notation and terms

The points

$$P = \{p_i \in \mathbf{R}^3\}, i \in \{1, \cdots, N\}$$

are sample from

the unknown surface S

the normal information may provided is represented as

$$N = \{n_i \in \mathbf{R}^3, \| \ n_i \ \| = 1\}$$

We want to define a surface $\hat{S}$ from the points P (and possibly N)
We call the reconstructed surface to be **interpolating** if

$$P \in \hat{S}$$

otherwise it's a **approximating.**

Only **approximation** is considered here:

- assume the surface is not too wiggly and contain some noise
- the result allows smoothing the noise and provide reasonably behaved surface

Notation :

$$p_i = (q_i, f_i)$$

where q_i is in parameter space(R^2 here), and $f_i = f(q_i)$.

Figure 4.4: A point set is (*a*) respectively approximated by a curve and (*b*) interpolated.

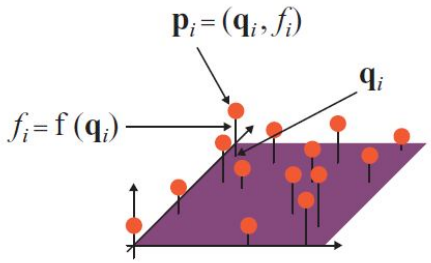


Figure 4.5: The notation for the functional (parameterized) setting.

4.2.3 Interpolation and approximation of functional data

Goal : determine a f interpolate or approximates p_i:

$$\hat{f} \approx f_i$$

A simple approach :

1 given x, find the closed p_i

2 set $\hat{f}(x) = f_i$

- interpolation, but not continuous

An obvious improvement :

combine the values of several close points:

$$\hat{f}(x) = \sum_i \omega_i(x) f_i$$

where $\omega_i(x)$ are weight functions on x.

Different choice of $\omega_i(x)$ generates different methods.

Radical Basis Functions

A general approach :

$$\omega_i(x) = \frac{c_i}{f_i}\theta(\parallel x - q_i \parallel) \iff \omega_i(x) f_i = c_i\theta(\parallel x - q_i \parallel)$$

where θ describe the influence of **distance** .

the c_i here is defined by interpolation :

$$\hat{f}(q_j) = \sum_i c_i\theta(\parallel q_j - q_i \parallel) = f_j$$

which is in fact :

$$\begin{pmatrix} \theta(\parallel q_0 - q_0 \parallel) & \theta(\parallel q_0 - q_1 \parallel) & \theta(\parallel q_0 - q_2 \parallel) & \cdots \\ \theta(\parallel q_1 - q_0 \parallel) & \theta(\parallel q_1 - q_1 \parallel) & \theta(\parallel q_1 - q_2 \parallel) & \cdots \\ \theta(\parallel q_2 - q_0 \parallel) & \theta(\parallel q_2 - q_1 \parallel) & \theta(\parallel q_2 - q_2 \parallel) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \end{pmatrix}$$

Standard choices are

$$\theta(\delta) = \delta^{-u},\ u \in \mathbf{N}$$

or the Gaussian

$$\theta(\delta) = exp(\delta^2/h^2)$$

Notation : these functions are **impractical** *since*

- *each point influences every other point*
- *a global dense linear system in any point*

Change

- *use locally supported radical functions, with far distance terms vanishes*
- *lead to sparse linear systems*

Least Squares Fitting

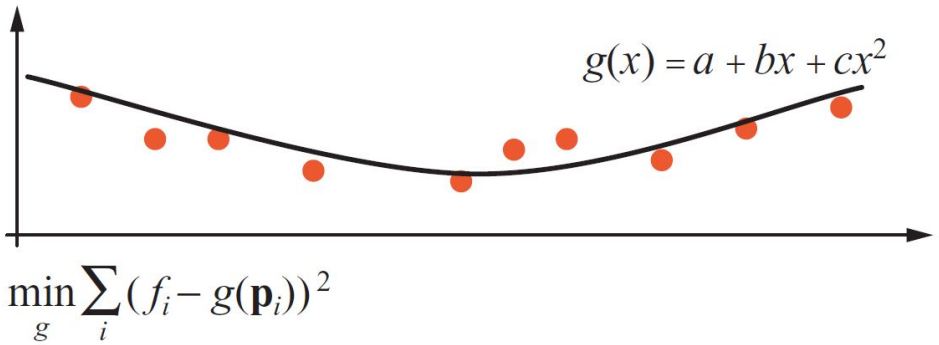


Figure 4.7: Least squares: Fitting a (here: quadratic) function $\hat{f}$ by minimizing the squares of differences between the given function values f_i and the values of the quadratic function at the corresponding locations $\mathbf{q}_i$. The standard approach is to minimize among all quadratic functions $\hat{f}(x) = a + bx + cx^2$ (i.e., $\min_{a,b,c} \sum_i (f_i - \hat{f}(\mathbf{q}_i))^2$); here we rediscover the same solution starting from different assumptions (see text).

we represent $\hat{f}$:

$$\hat{f}(x) = \sum_i \omega_i(x) f_i$$

and We ask that $\hat{f}$ has a certain **precision** :

- $\hat{f}$ is exact for each function g in a precision set G

$$g(x) = \sum_i \omega_i(x) g(q_i), \ \forall g \in G$$

For example, the set quadratic polynomials $g(x) = a + b^T x + x^T C x$, we then have:

$$\begin{aligned} 1 &= \sum_i \omega_i(x) \cdot 1 \\ x_0 &= \sum_i \omega_i(x) \cdot q_{i_0} \\ &\vdots \\ x_0^2 &= \sum_i \omega_i(x) \cdot q_{i_0}^2 \\ &\vdots \end{aligned}$$

Then the requirement of the precision set becomes :

$$a + b^T x + x^T C x = \sum_i \omega_i(x)(a + b^T q_i + q_i^T C q_i)$$

or

$$QW(x) = Z$$

Since we have **more points** than dimensions of the precision set, the **weights** have to be **further restricted**.
We take :

$$\min_{\{\omega_i(x)\}} \sum (w_i(x))^2 = \min_{W(x)} W(x)^T W(x)$$

How to find this minimum ?

assume we know the solution $W(x)$

we can find a coefficient vector $(a, b_0, \cdots)$, minimizing

$$W(x)^T W(x) - (a, b_0, \cdots) Q W(x)$$

use the first order necessary condition, we have:

$$W(x)^T = Q^T (a, b_0, \cdots)^T$$

substitute in to $QW(x) = Z$, we have

$$Q Q^T (a, b_0, \cdots)^T = Z$$

solving the linear system and we obtain W(x)

Moving Least Squares

the only modification is that, we localize wights:

$$\min_{w_i(x)} \sum_i w_i^2(x)\eta(\| q_i - x \|) = \min_{W(x)} W(x)^T E(x) W(x)$$

where $\eta(\| q_i - x \|)$ *penalizes the influence of points far away from* x.

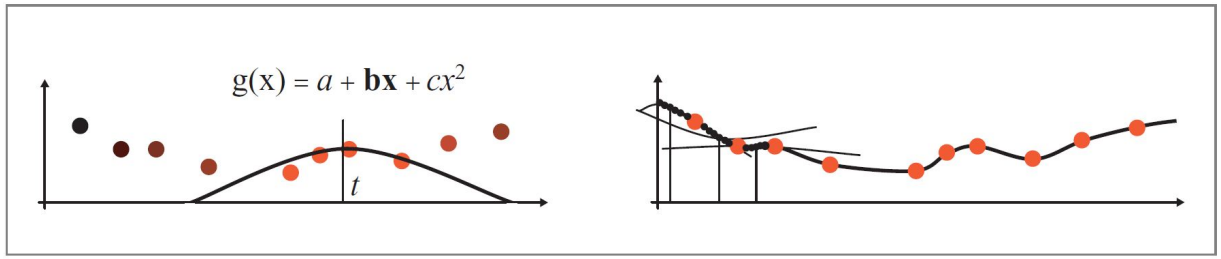


Figure 4.8: Moving least squares: In each location **x** a polynomial is computed using the least squares method, however, weighting the influence of the data points based on their distance. The value of this polynomial at **x** yields the functional approximation (*left*). The set of locally approximated function values, together, forms the approximated curve (*right*).

4.2.4 Normals

Problems with **NO** suitable parameter domain

approach : approximating *tangent planes or normals* .

If normals are not part of input, they are estimated:

- find $N_k(q)$
- search a plane $H(x) : n^T q = n^T p_i$, which minimize

$$(n^T(q - p_i))^2$$

- take the normal of $H(x)$

or using a **locally supported weight function θ, and minimize :**

$$\min_{\|n\|=1} \sum_i (n^T(p_i - q))^2 \theta(\| p_i - q \|)$$

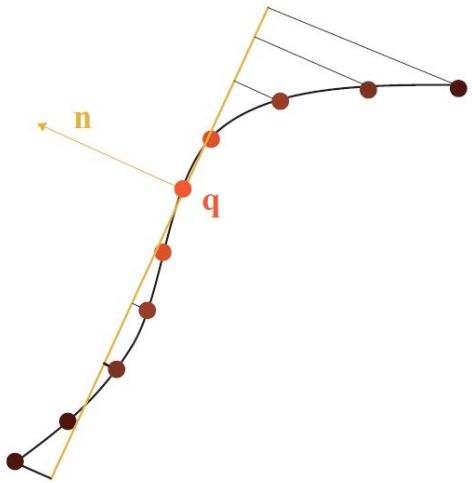


Figure 4.9: Estimating the normal direction close to a point $\mathbf{q}$: A unit normal $\mathbf{n}$ is computed so that the plane orthogonal to $\mathbf{n}$ minimizes the squared distances to the points. The extra constraint $\|\mathbf{n}\| = 1$ makes this a nonlinear problem that can be solved, however, by an eigenvalue/eigenvector computation.

4.2.5 Implicit surfaces from points and offset points

Implicit form :

$$S = \{x | \hat{f}(p_i) = 0\}$$

is a set of constraints :

$$\hat{f}(p_i) = 0$$

- additional **nonzero** constraints should be added, or the solution is $\hat{f} \equiv 0$

A standard trick : **moving a small step along the normal(say δ) and set**

$$\hat{f}(p_i + \delta n_i) = \delta$$

4.2.6 Implicit surface from points and tangent frames

Idea :

- estimate local tangent frames
- use a standard technique in this local tangent frame

MLS Surfaces

- compute a locally tangent reference domain H
- a local bivariate polynomial is fitted over H

The local reference domain $H = \{x| < n, x > -D = 0\}$ is determined by minimizing:

$$\sum_{i=1}^{N}(\langle n, p_i - r - tn\rangle)^2\theta(\| p_i - r - tn \|)$$

among all normal n and offset t.

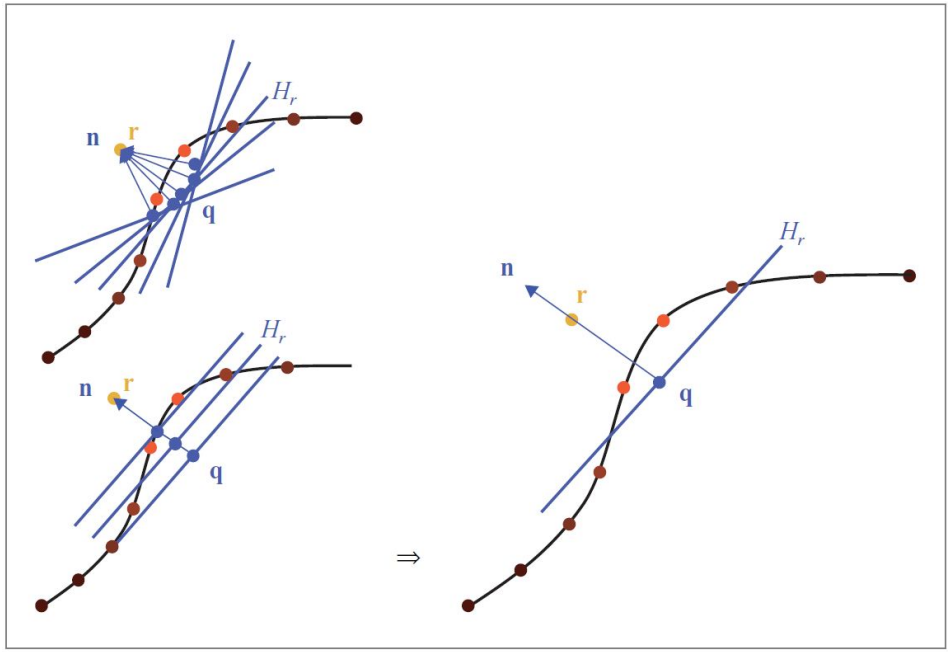


Figure 4.12: The reference plane for the first step of the MLS projection is found by optimizing over all normal directions **n** and all offsets *t*.

let q_i be the projection of p_i onto H, and f_i the height of p_i over H,thus

$$f_i = n \cdot (p_i - q)$$

then obtain the approximation g by minimizing the least square error:

$$\sum_i^N (g(x_i, y_i) - f_i)^2 \theta(\| p_i - r - tn \|)$$

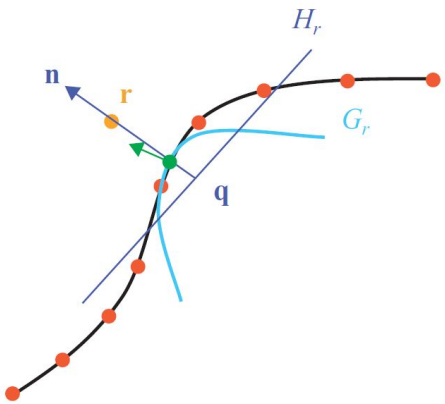


Figure 4.13: The reference plane is then used to compute a polynomial least squares approximation. The value at the origin of the reference frame is used as the projection of **q**.

Thus, the projection of r is

$$MLS(r) = r + (t + g(0,0))n$$

Formally, the surface S_P is the set of points that projects to itself :

$$\hat{f}(x) = \| (t + g(0,0))n(x) \|$$

Surfaces from Normals and Weighted Averages

Inspired by MLS surfaces, and simplify the nonlinear optimization

1. compute a tangent frame in x
2. approximate the data with a locally weighted least squares polynomial
3. intersection of the normal and polynomial is x'
4. repeat

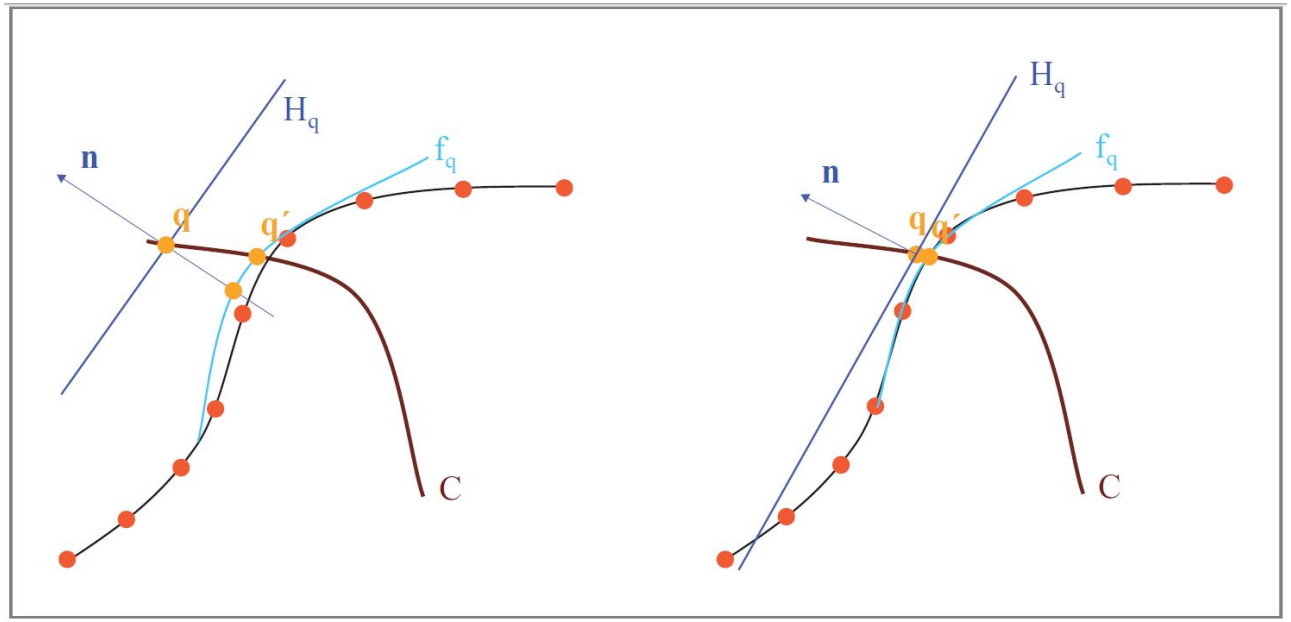


Figure 4.14: An alternative constructive definition of a projection. Reference planes are computed through $\mathbf{x}$ yielding the next point $\mathbf{x}'$. The process is repeated until convergence.

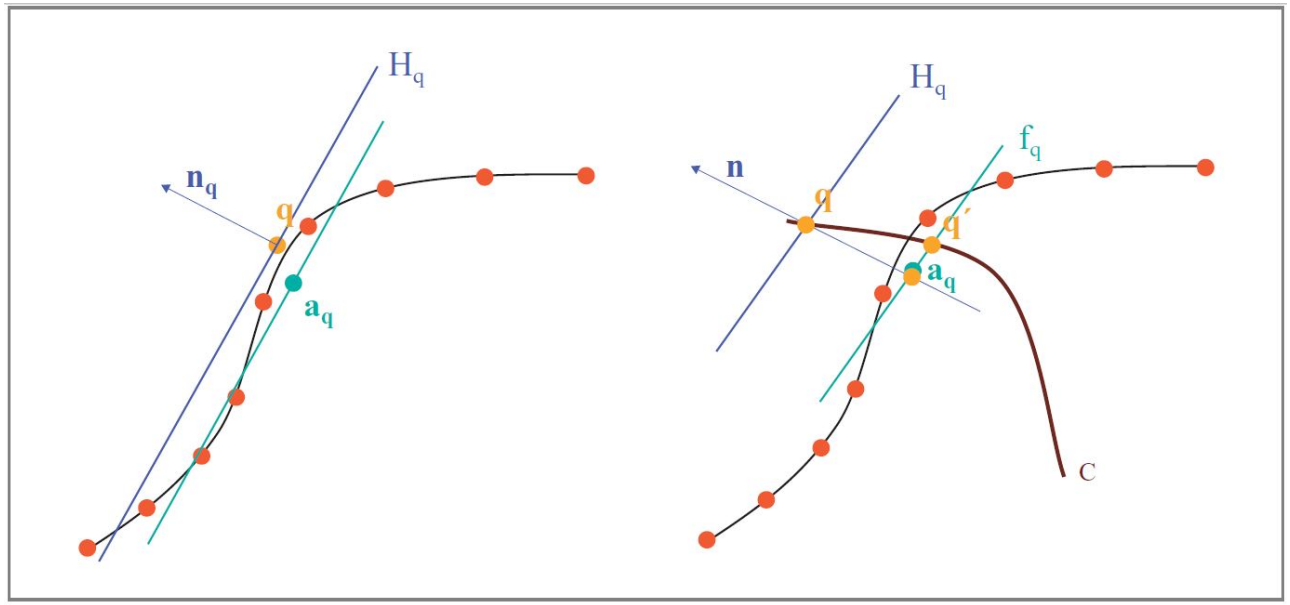


Figure 4.15: If constant polynomial approximations are used, the approximation passes through the locally weighted average of points $\mathbf{a}(\mathbf{q})$. A single step in the iteration then becomes $\mathbf{q}' = \mathbf{q} - \mathbf{n}(\mathbf{q})^{\mathsf{T}}(\mathbf{q} - \mathbf{a}(\mathbf{q}))\,\mathbf{n}(\mathbf{q})$, and repeating this assignment is the recommended projection operator. The stationary points of this operator also give rise to an interpretation of the surface defined by the implicit function $f(\mathbf{x}) = \mathbf{n}(\mathbf{x})(\mathbf{a}(\mathbf{x}) - \mathbf{x}) = 0$.

4.3 Sampling of Point Models

- 4.3.1 Overview
- 4.3.2 Decimation and resampling techniques
- 4.3.3 Analysis and comparison

4.3.1 Overview

point-based surface representation :

$$\text{discrete point space} \longrightarrow \text{continuous surface}$$

Question : what density is needed to **capture the relevant geometric details** ?

Since the point models are usually obtained from laser scanning, which leads to rather **dense** point clouds, much more attention is attached to **Downsampling** methods than **Upsampling** methods.

point-based presentations :

- error $\times\frac{1}{2} \Longleftarrow$ point samples $\times 4$

In order to fill in the gap, **splat-based representations** are often used.

splat-based representations

- surface approximated by little disk or ellipse
- provide 1^{st} approximation to the surface
- error $\times\frac{1}{2} \Longleftarrow$ splat number $\times 2$

4.3.2 Decimation and resampling techniques

Point-simplification Methods

- Clustering
- Iterative
- Particle simulation

Clustering Methods

Idea : group the input into patches that **do not** exceed a given upper bound for **size** (*diameter*) or **variation** (*normal or through covariance analysis*).

- incremental approach : down-top, starting from random seeds, and grow by adding neighbors
- hierarchical approach : top-down, splitting the point cloud, through a covariance analysis

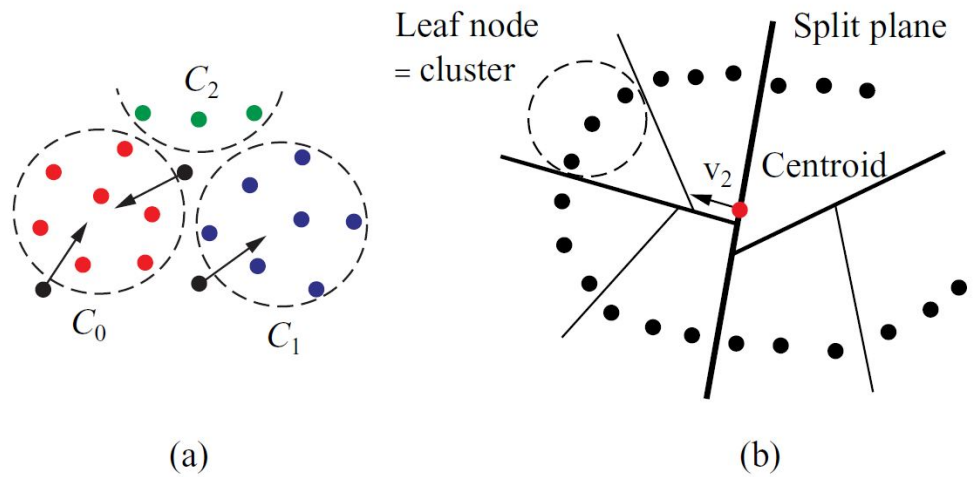


Figure 4.19: (*a*) Clustering by incremental region growing, where "stray samples" (black dots) are attached to the cluster with closest centroid. (*b*) Hierarchical clustering, where the thickness of the lines indicates the level of the BSP tree (2D for illustration).

Iterative methods : *reduce the points using a sequence of decimation operators.*

- operators arranged by the error it caused
- smallest error, first applied
- k-nearest neighbor relation needed
- tangent plane E_i is estimated for error estimation for each p

Particle simulation methods

- desired number of particles randomly spread across the surface
- equalize their position using point repulsion algorithm
- add samples in lower density region **while** ensure uniformity
- point movement restricted on the surface

the three methods above:

- cannot take an a priori approximation error tolerance into account
- pure greedy simplification produces produces nonuniform sampling density

Splat-Decimation Methods

considering the whole splat geometry, the resulting sampling quality can be largely improved

Procedure:

- initial splats created by hierarchical method
- a sequence of **splat-merge operator** are arranged by error it caused
- apply operator, and update operator priority in the queue

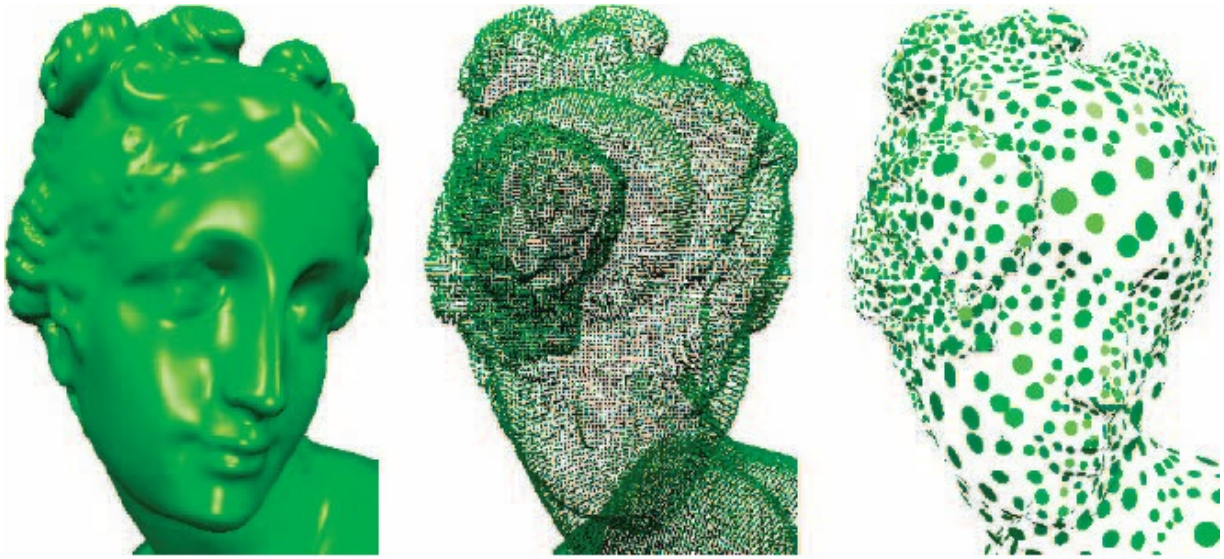


Figure 4.21: The original point model (left and middle, 352,000 points) is decimated to 30,000 circular surface splats (right).

splat s_i :

- 3D ellipse
- center c_i
- unit normal n_i
- two additional nonuniform vectors u_i and v_i, the major and minor axes

initial s_i of p_i :

- $N_k(p_i)$ is needed $\Rightarrow$ tangent plane $H \Rightarrow n_i$
- $c_i = p_i$
- u_i and v_i : any orthogonal vectors parallel to H with length r_i

$$r_i = \max_j \parallel (p_j - c_i) - n_i^T (p_j - c_i) n_i \parallel$$

for all $p_j \in N_k(p_i)$

to utilize the full geometry, **two different error metrics** are also generalized

the L^2 error metric

the error of merging s_l and s_r to s_m is

$$\epsilon_\Psi = \| e \| \cdot \sum_{f \in \{f_m\}} |dist(p_f, s_m)|^2, \ \{f_m\} = \{f_l\} \cup \{f_r\}$$

where $\{f_i\}$ are indices of points in $\{s_i\}$.

Apply **principle component analysis** to $P_m = \{p_f\}, f \in \{f_m\}$

get first 3 eigenvectors e_1, e_2, e_3 of the first 3 eigenvalue $\lambda_1 > \lambda_2 > \lambda_3$

$u_m = e_1, v_m = e_2, n_m = e_3$

$c_m = \bar{p}$, the average

the $L^{2,1}$ error metric

- the area of two splats to be merged are needed : $|s_l|, |s_r|$

$$\epsilon_\Phi = \| e \| \cdot (|s_l| + |s_r|) \cdot \| n_l - n_r \|^2$$

and

$$c_m = \frac{|s_l| \cdot c_l + |s_r| \cdot c_r}{|s_l| + |s_r|}$$

$$n_m = \frac{|s_l| \cdot n_l + |s_r| \cdot n_r}{|s_l| + |s_r|}$$

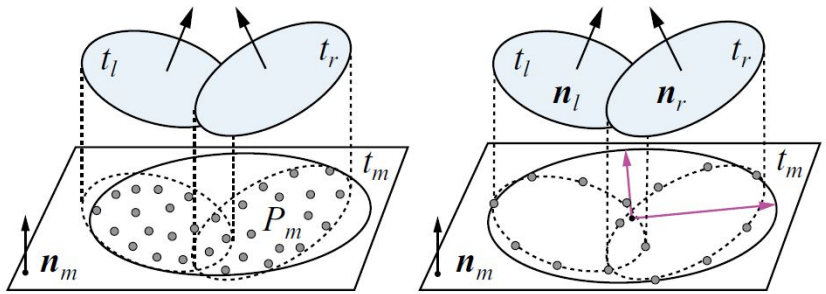


Figure 4.23: Splat merge operators according to L^2 error metric (***left***) and $L^{2,1}$ metric (***right***), which will merge splats t_l and t_r into a new splat t_m.

the techniques of Wu and Kobbelt(OSS)

in this algorithm, splat subsampling problem is formulated into a **minimum dominating set problem**

a global optimization is applied to compute the **minimal number** of splats for a sample p_i,

- compute its distance to the "splat set" T
- obtain a coverage relation $C_\epsilon \subset P \times T$
- the patch $Q_j = C_\epsilon(*, s_j)$ correspond to s_j
- the problem then formulated as a minimum dominating set problem

4.3.3 Analysis and comparison

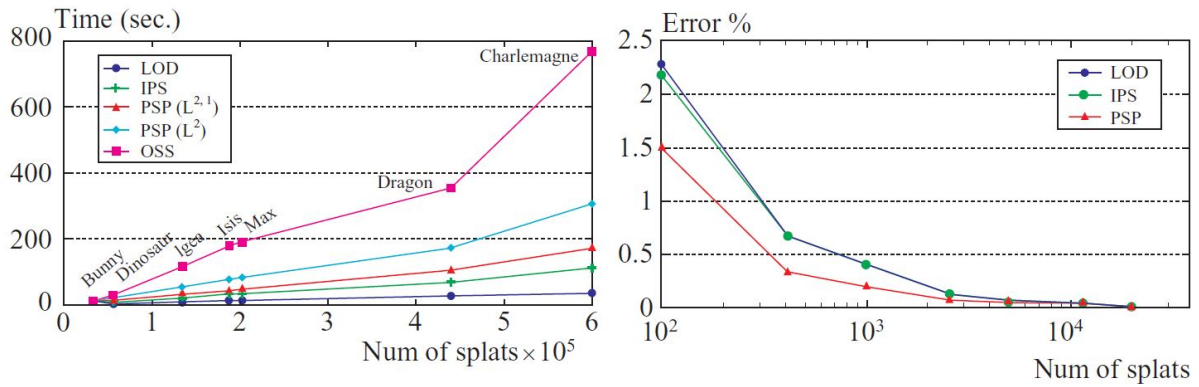


Figure 4.29: (*Left*) Computation times for different point decimation methods where for PSP, OSS, and IPS, times are measured for a simplification to 1% of the input model size and LOD is its whole structure creation time. (*Right*) Error comparisons on bunny model (35,000 points) for three different progressive splat decimators.

Figure 4.28: Bunny model (see also Figure 4.30) decimated to similar number of splats by single-resolution OSS (left, 2,577) and progressive PSP (right, 2,591) algorithms. Although PSP and OSS have quite close errors (0.103% to 0.092%), being able to concentrate more splats on regions of high curvatures, OSS gives better splat shapes and distribution than PSP.

Table 4.1: Area Ratio for Different Point Decimation Methods, Normalized to the Initial Surface Area of the Triangle Mesh. This Factor Measures How Much Overdraw Occurs in the Rasterization of the Splats.

n_{splat}	PSP	LOD	IPS	OSS
415	2.18	2.34	4.14	1.63
2,591	2.52	2.71	4.12	2.15
11,588	3.23	3.22	4.23	3.13

- LOD merely adopts the octree space-partitioning scheme
- IPS only considers splat centers

without whole splats geometry, they could not produce as promising results as PSP.

- OSS produces best quality due to its global optimization
- PSP method comes quite close to the best OSS solution

Conclusions

- point models are usually highly oversampled
- for processing efficiency, **subsampling or decimation** are important

- methods decimation using points only **approximate geometry weakly**
- splat-based representation are a powerful extension

4.4 Efficient data structures

content

- 4.4.1 Overview
- 4.4.2 Spatial data organization

When very large data volumes must be managed and processed, **efficient data organization and access methods** has to be considered carefully:

1 What classes of query requests to retrieve points must be supported ?

2 What type of storage constraints are imposed to represent point splats ?

3 What are the requirements for dynamic point insertions and deletions ?

we ignore question 3 as the point data to be visualized is static

4.4.2 Spatial data organization

the goal :

- index the space : decompose it into **cells**, provide a **mapping** between these and the space occupied by an object

a straight method is partition the data into *buckets* :

$$B_j = \{p_{1_j}, \cdots, p_{n_j}\},\ P = \bigcup_{j=1}^{m} B_j$$

and the bounding attributes of B_j can be represent by *mass center*

$$\hat{p}_j = 1/n_j \sum p_{i_j}$$

and bounding radius

$$\hat{r}_j = \max |\hat{p}_j - p_{i_j}| + r_{i_j}$$

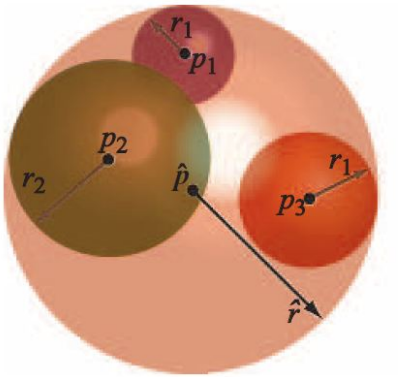


Figure : Illustration of a bounding sphere

However, for large point set P, a bucketization

$$P = \bigcup_{j=1}^{m} B_j$$

may result in **large** m , which may in turn have to be organized.

Most spatial data structures is constructed as follows:

- employ some sort of **subdivision of space**
- **leaf nodes** becomes the actual data buckets

Notice : it's important to avoid excessive recursive subdivision to a **single** data element, but strive to a bucket of **k** points.

Octrees : *one of the most common choices*

Procedure:

1. *start with a bounding box cell*
2. *each cell is recursively subdivided into eight nonempty octants*
3. *terminate as a* **leaf** when a cell have **less than** k points

There are also two main strategies:

- *binary subdivision of all dimension – region octree : simpler structure*
- *subdivision at a point inside the cell – point octree : more adaptive to distribution*

Algorithm :

Octree($\mathcal{P}$)
  if $|\mathcal{P}| \leq k$ **then**
    return New_leaf_node($\mathcal{P}$)

  $\mathbf{p}_{split} \leftarrow$ Get_new_split_position($\mathcal{P}$)
  for $i = 1$ **to** $|\mathcal{P}|$
    $r_{split} \leftarrow \text{MAX}(|\mathbf{p}_{split} - \mathbf{p}_i| + r_i, r_{split})$
    $j \leftarrow$ Get_octant($\mathbf{p}_{split}, \mathbf{p}_i$)
    $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup p_i$
  for $j = 1$ **to** 8
    if $\mathcal{P}_j \neq \emptyset$ **then**
      $c_j \leftarrow$ Octree($\mathcal{P}_j$)
  return New_octree_node($\mathbf{p}_{split}, r_{split}, c_{1\ldots8}$)

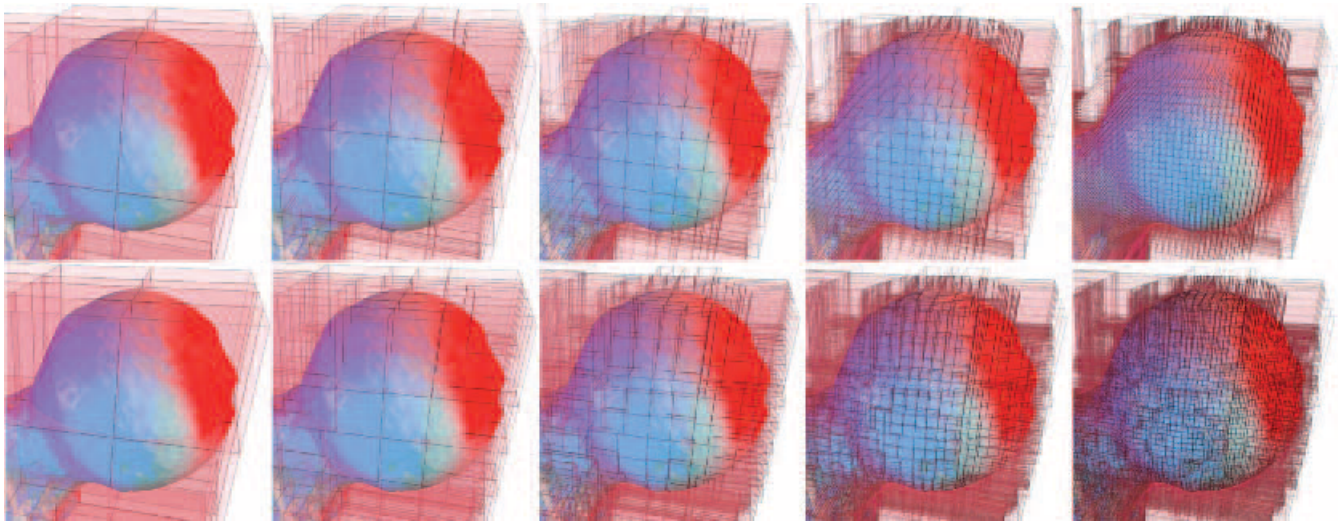


Figure 4.34: Examples of regular region octree subdivision, binary in each dimension (upper row), versus adaptive point octree subdivision at arbitrary split positions (lower row).

octrees : a simple hierarchical

but in general, points in 3D

- can not evenly be subdivided into eight octants
- lead to unbalanced and suboptimal structure

The K-d-trees

- split along **one** direction : the dimension with **largest spatial extent**
- 2 subregions have equal number of elements

Algorithm

need three indices arrays X, Y, Z : sorted indices of respective dimension

K-d-tree$(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$
  $m \leftarrow |\mathcal{X}|$
  if $m \leq k$ **then**
    return New_leaf_node$(\mathcal{P}, \mathcal{X}, m)$
$cutdim \leftarrow$ Dimension_of_largest_extent$(\mathcal{P}, \mathcal{X}, \mathcal{Y}, \mathcal{Z})$
$median \leftarrow$ **switch** $(cutdim)$
  $x : \mathcal{X}[m/2 + 1]$
  $y : \mathcal{Y}[m/2 + 1]$
  $z : \mathcal{Z}[m/2 + 1]$
$\mathbf{p}_{split} \leftarrow \mathcal{P}[median]_{xyz}$
$x1 \leftarrow x2 \leftarrow y1 \leftarrow y2 \leftarrow z1 \leftarrow z2 \leftarrow 1$
for $i = 1$ **to** m
  $r_{split} \leftarrow \text{MAX}(|\mathbf{p}_{split} - \mathcal{P}[\mathcal{X}[i]]_{xyz}| + \mathcal{P}[\mathcal{X}[i]]_r, r_{split})$
  if $\mathcal{P}[\mathcal{X}[i]]_{cutdim} < \mathbf{p}_{cutdim_{split}}$ **then**
    $\mathcal{X}1[x1] \leftarrow \mathcal{X}[i]\ ; x1 \leftarrow x1 + 1$
  else if $\mathcal{X}[i] \neq median$ **then**
    $\mathcal{X}2[x2] \leftarrow \mathcal{X}[i]\ ; x2 \leftarrow x2 + 1$

if $\mathcal{P}[\mathcal{Y}[i]]_{cutdim} < \mathbf{p}_{cutdim_{split}}$ **then**
  $\mathcal{Y}1[y1] \leftarrow \mathcal{Y}[i]\ ; y1 \leftarrow y1 + 1$
else if $\mathcal{Y}[i] \neq median$ **then**
  $\mathcal{Y}2[y2] \leftarrow \mathcal{Y}[i]\ ; y2 \leftarrow y2 + 1$
if $\mathcal{P}[\mathcal{Z}[i]]_{cutdim} < \mathbf{p}_{cutdim_{split}}$ **then**
  $\mathcal{Z}1[z1] \leftarrow \mathcal{Z}[i]\ ; z1 \leftarrow z1 + 1$
else if $\mathcal{Z}[i] \neq median$ **then**
  $\mathcal{Z}2[z2] \leftarrow \mathcal{Z}[i]\ ; z2 \leftarrow z2 + 1$

$left \leftarrow right \leftarrow$ NULL
if $m/2 > 0$ **then**
  $left \leftarrow$ K-d-tree $(\mathcal{P}, \mathcal{X}1, \mathcal{Y}1, \mathcal{Z}1)$
if $m/2 + 1 < m$ **then**
  $right \leftarrow$ K-d-tree $(\mathcal{P}, \mathcal{X}2, \mathcal{Y}2, \mathcal{Z}2)$
return New K-d-tree_node $(\mathbf{p}_{split}, r_{split}, left, right)$

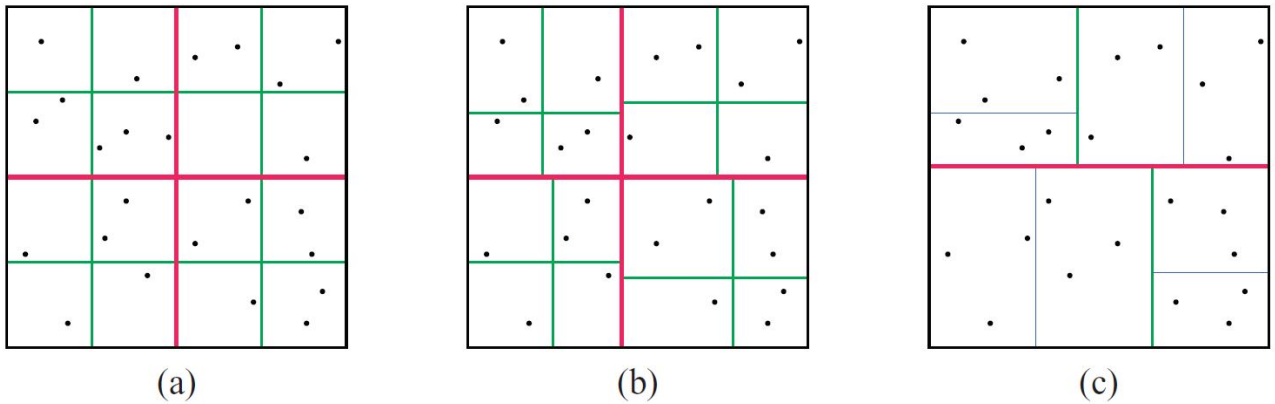


Figure 4.36: Comparison of a (*a*) regular region quadtree, (*b*) adaptive point quadtree, and (*c*) 2D-tree split organization, for a bucket size of three.

4.5 Real-time refinement

content

- 4.5.1 Overview
- 4.5.2 One-ring neighborhood selection
- 4.5.3 Refinement algorithm

4.5.1 Overview

- necessary for high-quality rendering, multiresolution processing, etc.
- unstable, for tedious selection of neighborhood, without connectivity information

generally, we need $N(p)$, then decide where to insert new points

4.5.2 One-ring neighborhood selection

- a critical step : decides the new points' position
- directly affect the robustness of the refinement algorithm

The choice of $N(p)$ here is performed in three steps:

1. coarse selection
2. geodesic projection
3. fuzzy BSP selection

Step 1 : Coarse selection

first, compute the Euclidean neighborhood $N_r(p)$ of p as the indices set:

$$N_r(p) = \{i | p_i \in P^l, p_i \neq p, \parallel p - p_i \parallel < r\}$$

then **reduce it** by several binary rules.

For instance, the **co-cone rule** *:* p_0 *and* p_1 *can be neighbors only if* $p_1(p_0)$ *is in the complement of the double cone(co-cone) of apex* $p_0(p_1)$*, axis* $n_0(n_1)$*, and angle* θ_{cocone}*:*

$$C_{cocone}(p_0, p_1) \iff \begin{aligned} & Cos^{-1}\left(\left| n_0^T \frac{p_1 - p_0}{\parallel p_1 - p_0 \parallel}\right|\right) < \theta_{cocone} \\ and \;\; & Cos^{-1}\left(\left| n_1^T \frac{p_1 - p_0}{\parallel p_1 - p_0 \parallel}\right|\right) < \theta_{cocone} \end{aligned}$$

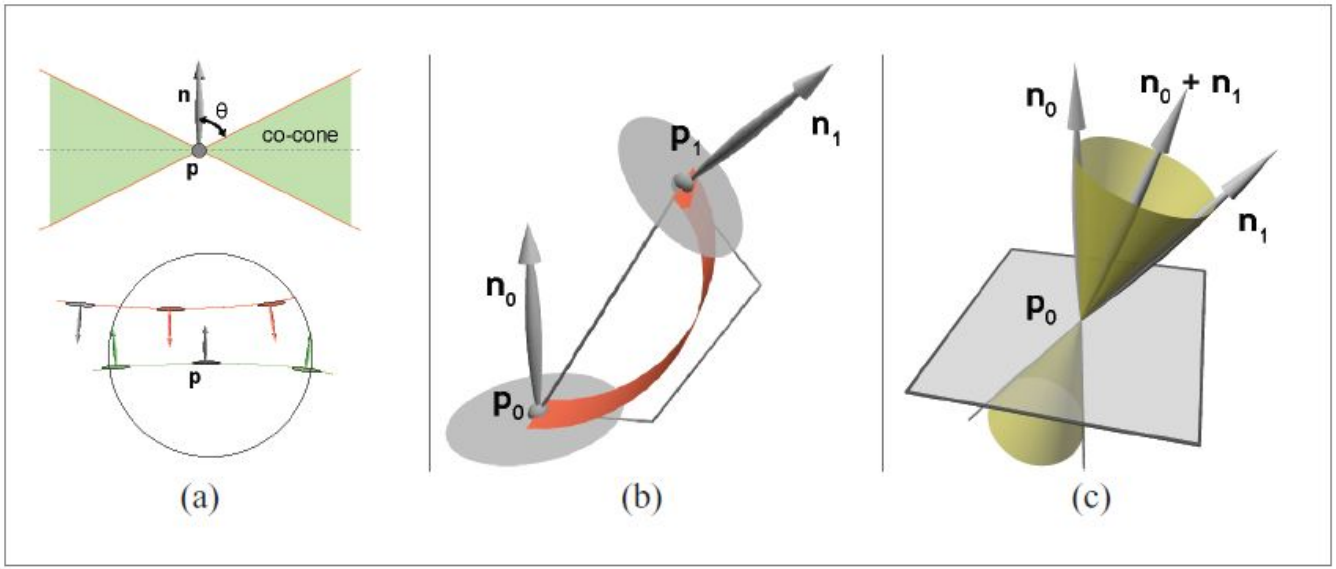


Figure 4.45: (*a*) *Top*: definition of a co-cone; *bottom*: a condition on the angle between normals can help to separate two close pieces of surface. (*b*) The relative positions and orientations of the points $\mathbf{p}_0$ and $\mathbf{p}_1$ are such that the construction of a Bézier curve by projection is inconsistent. (*c*) Given the position $\mathbf{p}_0$ and the two normals $\mathbf{n}_0$, $\mathbf{n}_1$, the point $\mathbf{p}_1$ must be outside the yellow cone.

a criterion between normals are also added:

$$C_{normal}(p_0, p_1) \iff Cos^{-1}(n_0^T n_1) < \theta_{normal}$$

which allow us to separate very close surfaces.

Then, $N_r(p)$ is reduced to:

$$\tilde{N}(p) = \{i \in N_r(p) | C_{cocone}(p_0, p_1) and C_{normal}(p_0, p_1) and \cdots\}$$

Step 2 : Geodesic Projection

since direct projection to the tangent plane significantly **reduces** *accuracy,* **geodesic projection** *is used here.*

procedure:

- *approximate geodesic distance of* p_0 *and* p_1 *by a cubic Bézier :* b_0, b_1, b_2, b_3
- *further approximate by the length of control polygon*

let $q(p_i, x)$ be the orthogonal projection operator, projecting x onto the tangent plane of p:

$$q(p_i, x) = x + n_i^T(p_i - x)n_i$$

and define $t_{i,j}$ be the pseudotangent vector from p_i toward p_j:

$$t_{i,j} = \frac{\| p_j - p_i \|}{3} \frac{q(p_i, p_j) - p_i}{\| q(p_i, p_j) - p_i \|}$$

obviously,

$$b_0 = p_0, b_1 = p_1$$

and b_1, b_2 are given by:

$$\begin{aligned} b_1 &= p_0 + t_{0,1} \\ b_2 &= p_1 + t_{1,0} \end{aligned}$$

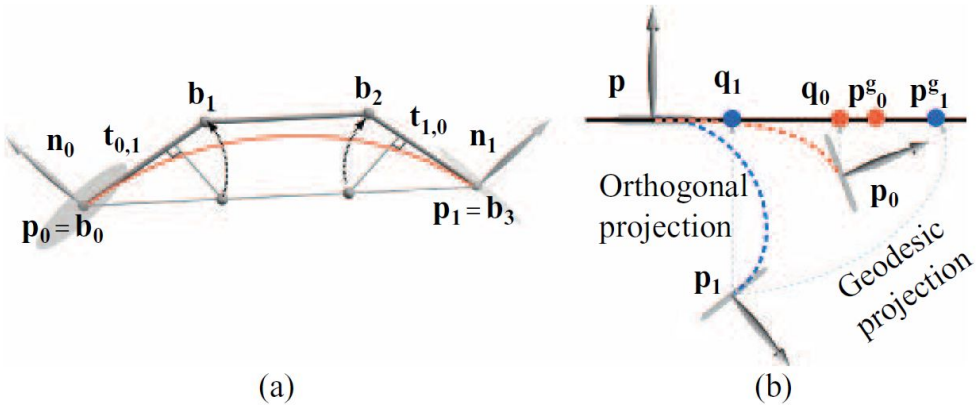


Figure 4.46: (*a*) Construction of a cubic Bézier curve interpolating two point normals. (*b*) Illustration of the geodesic projection against an orthogonal projection.

thus, the length of the control polygon, the approximated geodesic distance, is

$$\tilde{g}(p_0, p_1) = \frac{2}{3} \parallel p_1 - p_0 \parallel + \parallel b_2 - b_1 \parallel$$

and the **geodesic projection** of p_i is:

$$p_i^g = p + \tilde{g}(p, p_i) \frac{q(p, p_i) - p}{\parallel q(p, p_i) - p \parallel}$$

which allows a **correctly sorted** neighbors in the 2D domain, even in **high curvature** case.

Step 3 : Fuzzy BSP Selection

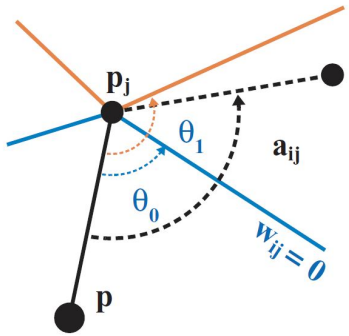
Intuition : remove points

- strongly ”behind” another
- slightly ”behind” two others

the notion ”behind” is described by ***badness*** ω_{ij} defined by some angles:

$$\omega_{ij} = \frac{\beta_{ij} - \theta_0}{\theta_1 - \theta_0} \approx \frac{cos(\beta_{ij}) - cos(\theta_0)}{cos(\theta_1) - cos(\theta_0)}$$

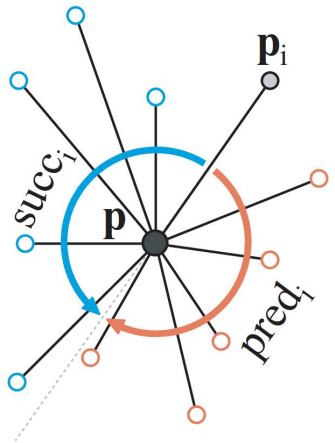
where $\beta_{ij} = \widehat{pp_j^g p_i^g}$ varies from θ_0 to θ_1



these fuzzy discriminant planes must be combined two by two.

Let

$$Succ_i = \{j \in \tilde{N}(p) | 0 < \widehat{p_i^g p p_j^g} < \pi\}$$
$$Pred_i = \{j \in \tilde{N}(p) | -\pi < \widehat{p_i^g p p_j^g} < 0\}$$



for each neighbor p_i of p, we compute:

$$\omega_{ij} = \max_{j \in Succ_i}(\omega_{ij}) + \max_{j \in Pred_i}(\omega_{ij})$$

and finally, the **one-ring neighborhood** $N(p)$ is chosen as:

$$N(p) = \{i | i \in \tilde{N}(p), \omega_i < 1\}$$

4.5.3 Refinement algorithm

for a triangular patch, we want the new point inserted **near the center**

- cubic Bézier triangular patch is used

Obtaining the *control points* :

1. 3 extremities $b_{300}, b_{030}, b_{003}$ are p_0, p_1, p_2
2. 6 boundary control points : depend on corresponding 2 extremities(as in previous section), for instance

$$b_{210} = p_0 + \frac{\| p_1 - p_0 \|}{3} \frac{q(p_0, p_1) - p_0}{\| q(p_0, p_1) - p_0 \|} = p_0 + t_{0,1}$$

3. central point

$$b_{111} = c + \frac{3}{2}(a - c)$$

where c is the **gravity** of p_0, p_1, p_2, and a is the **average of 6 boundary control points** .

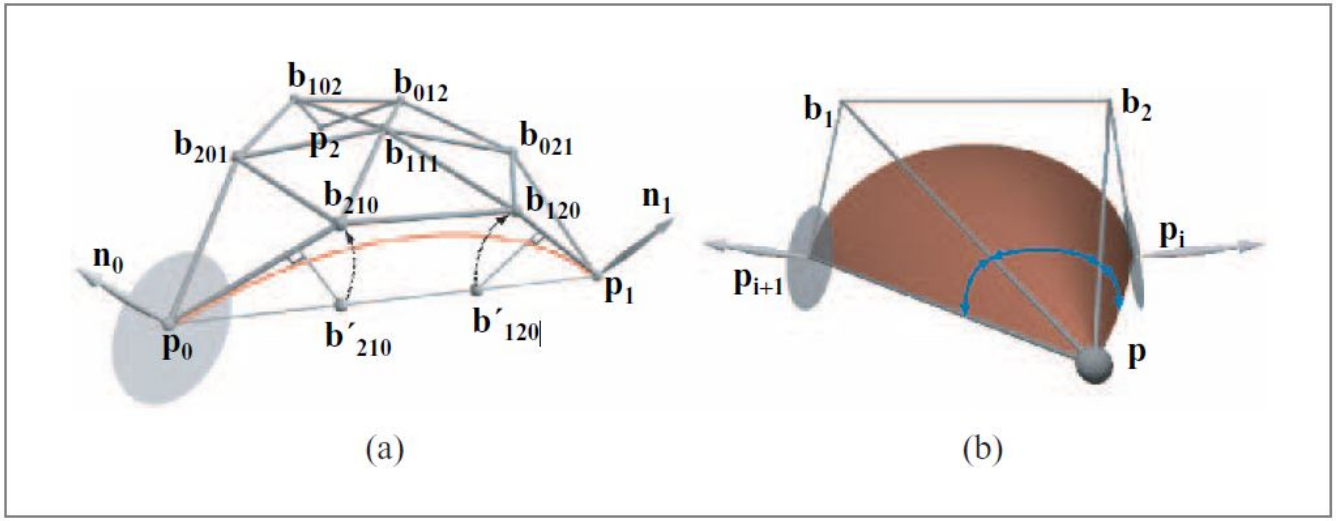


Figure 4.49: (*a*) Construction of the control polygon of a cubic Bézier triangle interpolating three splats. (*b*) The "curved angle" between two point normals $\mathbf{p}_i$, $\mathbf{p}_{i+1}$ relative to a third point $\mathbf{p}$ is specially useful for areas of high curvature. On this example there is a ratio of two between the geometric angle and the "curved angle."

the **position** of the new point is given by:

$$p_{new} = \frac{1}{3}(p_0 + p_1 + p_2) + \phi(p_0, p_1, p_2)$$

where ϕ is the *smoothing operator* :

$$\phi(p_0, p_1, p_2) = \frac{1}{6}\sum_{i=0}^{2} t_{i,i+1} + t_{i,i+2}$$

the **normal** is also estimated by the cross product of two *tangent vectors:*

$$\begin{aligned}
\frac{\partial B}{\partial u}(\frac{1}{3}, \frac{1}{3}) &= 7(p_1 - p_0) + b_{120} - b_{102} + b_{012} - b_{210} + 2(b_{021} - b_{201}) \\
\frac{\partial B}{\partial v}(\frac{1}{3}, \frac{1}{3}) &= 7(p_2 - p_0) + b_{102} - b_{120} + b_{021} - b_{201} + 2(b_{012} - b_{210})
\end{aligned}$$

Sampling control

- avoid oversampling : needn't insert in each triangle
- relevant new points : optimize the uniformity of the neighborhood

thus, criterions are needed to decide **whether a new point should be inserted** , or to **form a new neighborhood** $N'(p)$ **of** p in the refinement step.

Let

- $Y(p) = \{i | p_i \in P^{l+1}, \lambda r < \| p_i - p \| < r\}$ be points already inserted, **sufficiently close but not enough to be in** $N'(p)$
- $D(p) = \{x | i \in Y(p), \| x - p_i \| < \frac{1}{2}\lambda r\}$ be the **discard space** avoiding oversampling.
- $L(p) = \{\frac{1}{3}(p_0 + p_1 + p_2) + \phi(p_0, p_1, p_2) | i \in N(p)\}$ be **all possible new points** .

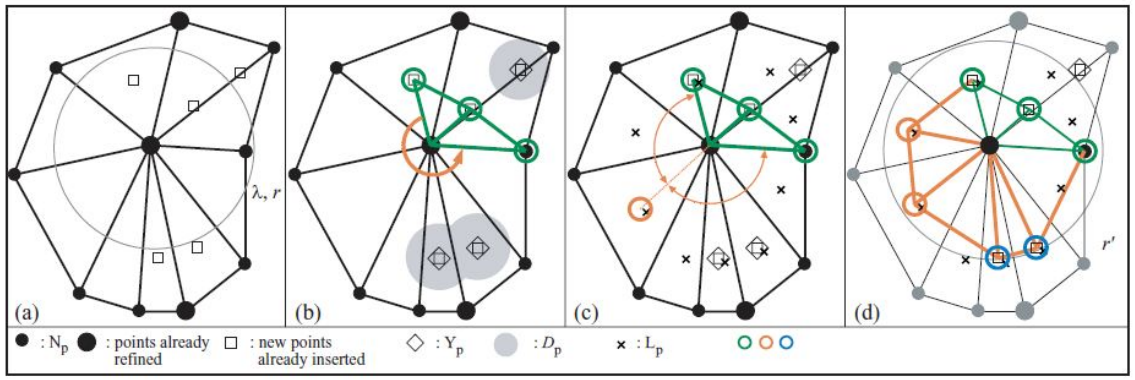
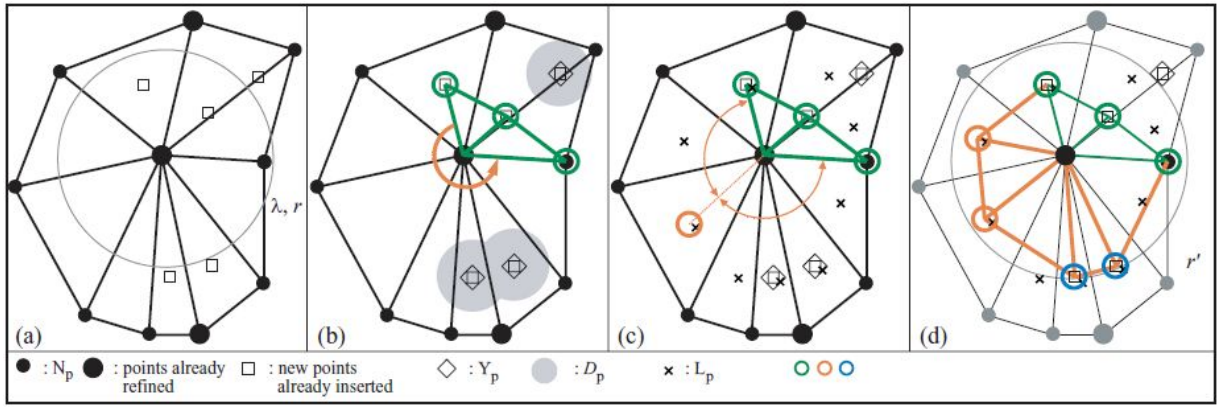


Figure 4.50: Local refinement of the current point $\mathbf{p}$. (*a*) The neighborhood of the point $\mathbf{p}$ before its own refinement. Two of its neighbors have already been refined: new points are represented by squares. (*b*) Initialization of the new neighborhood $\mathcal{N}'(\mathbf{p})$ (in green). (*c*) Selection of a point to insert. (*d*) The new neighborhood $\mathcal{N}'(\mathbf{p})$ is complete.

The insertion procedure is as following :

1. select p_j, $p_{j+1} \in N'(p)$ forming the **maximal angle**
2. select point in $L(p)$ **best balances** the point sampling (a good candidate : p_k such that the minimal of $\widehat{p_j p p_k}$ and $\widehat{p_k p p_{j+1}}$ is maximal)
3. if $p_k \notin D(p)$, insert it into P^{l+1} and $N'(p)$; else **no** new point inserted, the *closest one* in $Y(p)$ is inserted to $N'(p)$

thus, if the sample are locally dense enough, no new point is inserted.



Thanks!