Smoothing

Paolo Cignoni

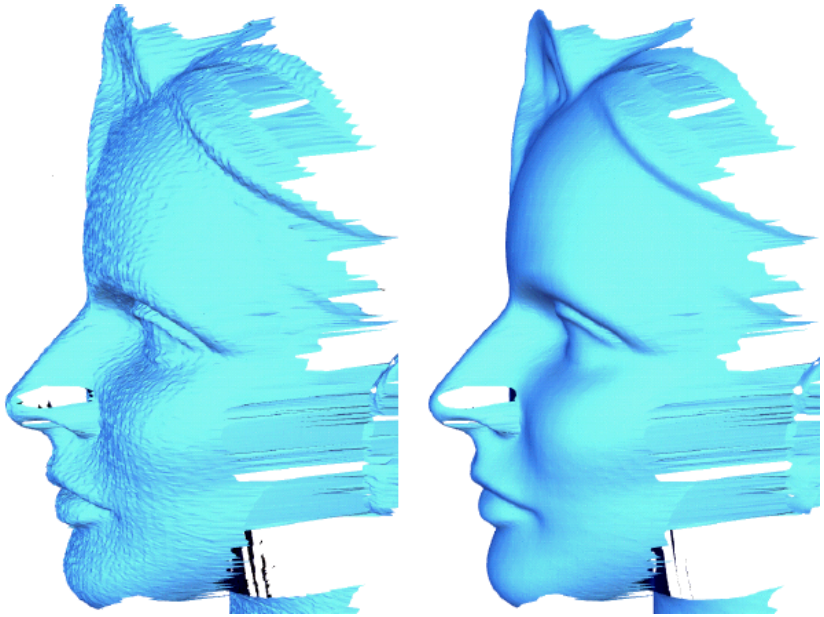
3D GEOMETRIC MODELING & PROCESSING

Smoothing

- Filtering out Noise from a mesh
- As in image processing

Motivations

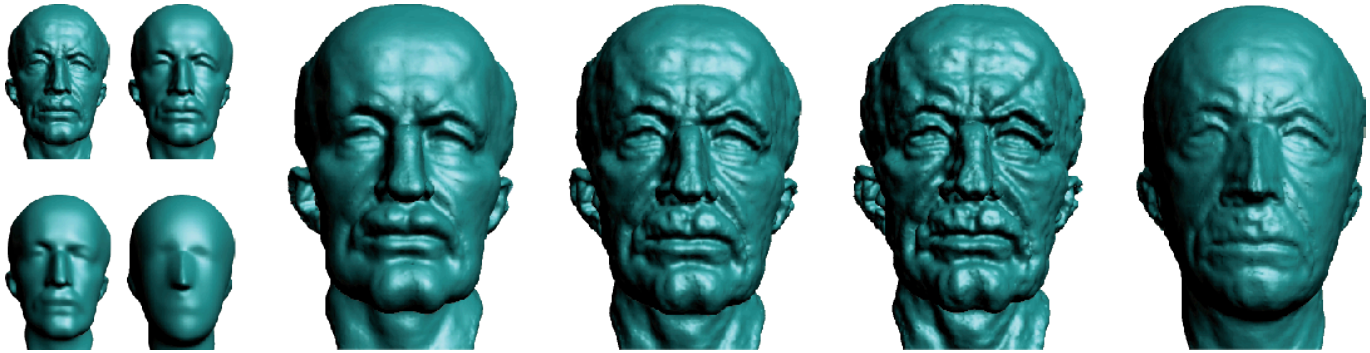
- Filter out high frequency components for noise removal



Desbrun, Meyer, Schroeder, Barr: *Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow*, SIGGRAPH 99

Motivations

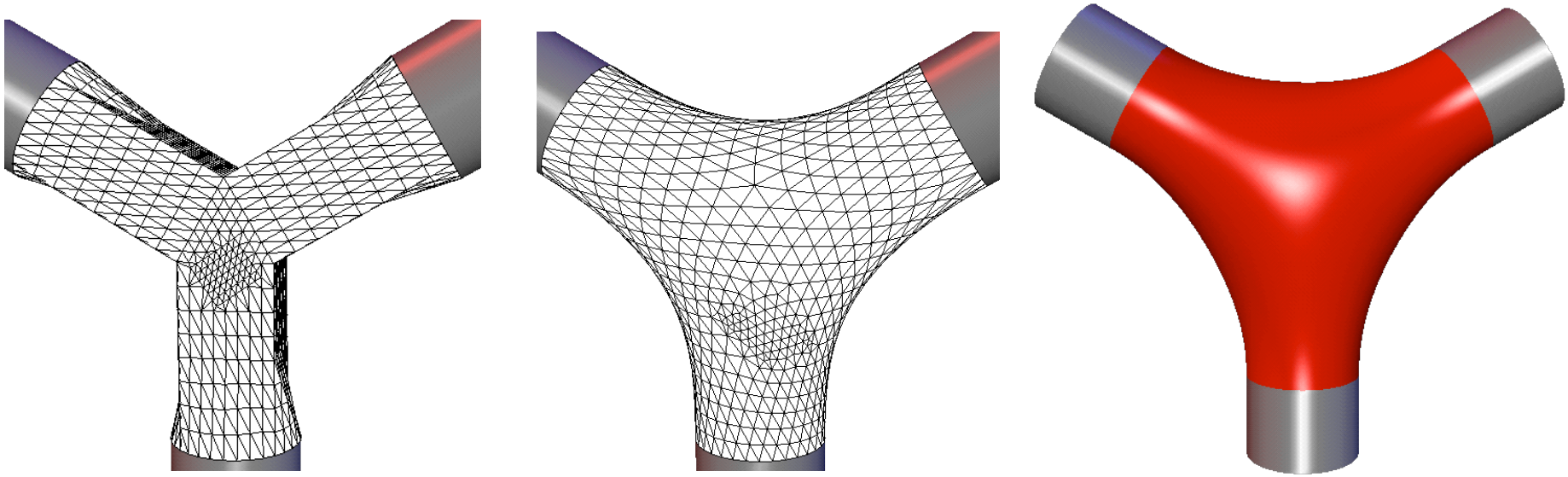
- Advanced Filtering / Signal Processing, Multiscale



Pauly, Kobbelt, Gross: *Point-Based Multi-Scale Surface Representation*, ACM TOG 2006

Motivations

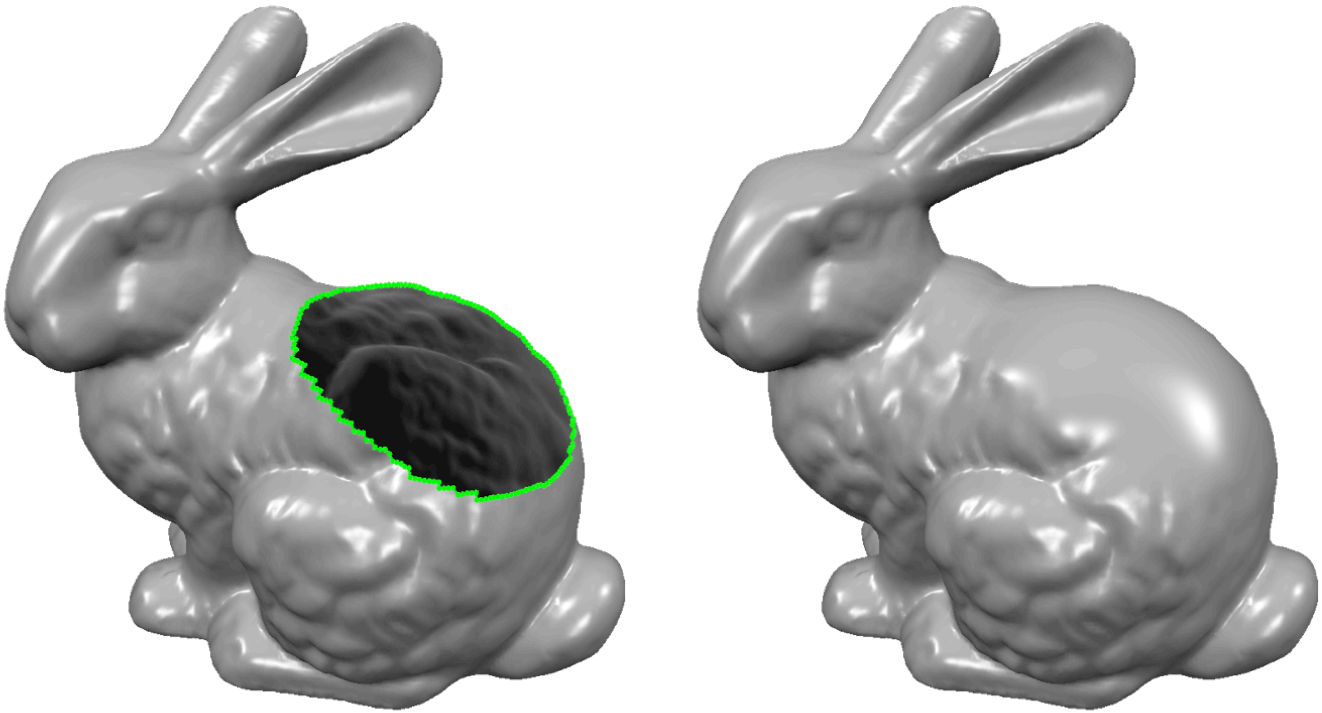
- Smooth surface design



Desbrun, Meyer, Schroeder, Barr: *Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow*, SIGGRAPH 99

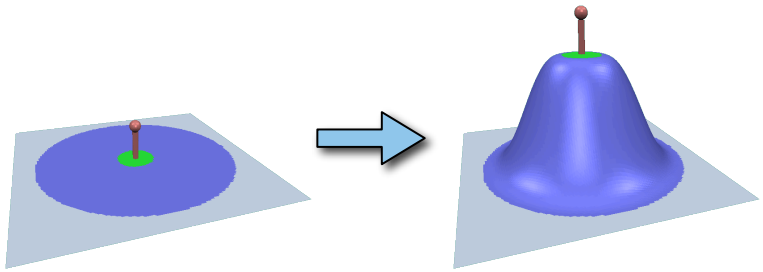
Motivations

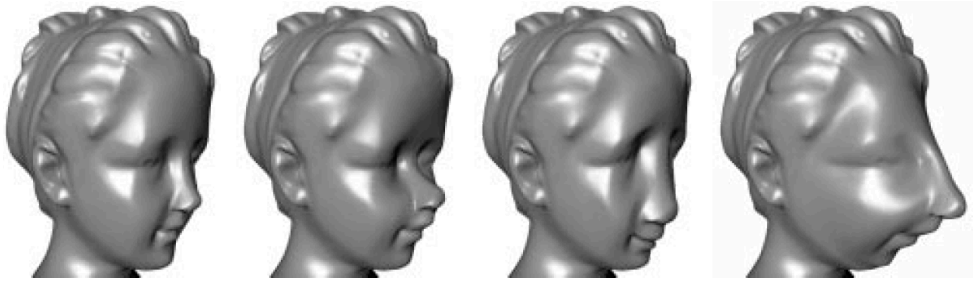
- Hole-filling with energy-minimizing patches



Motivations

- Mesh deformation
  - Smoothing terms are often needed to obtain nice deformations





noise and smoothing

- The mesh can be viewed as a two-dimensional vector signal defined over a manifold:

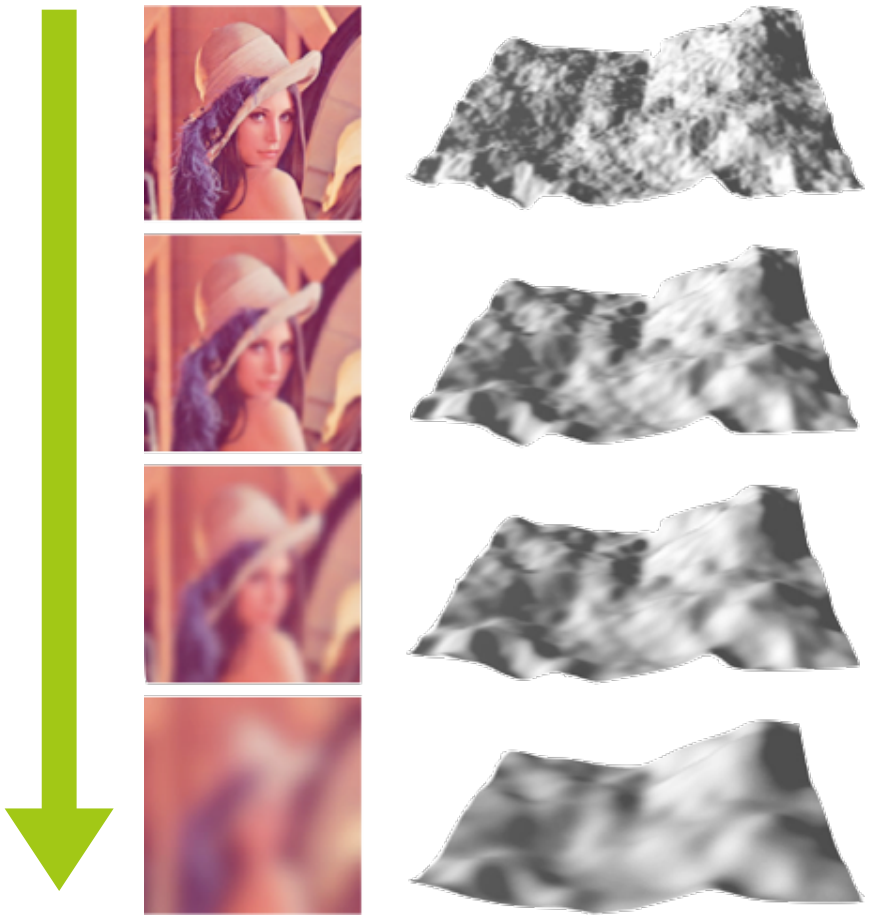
$$f : M \longrightarrow \mathbb{R}^3 \qquad f(v_i) = \mathbf{x}_i$$

  - where $\mathbf{x}_i$ denotes the position of vertex v_i in space and f is extended by linear interpolation to the rest of the mesh

- Under a signal-processing point of view, smoothing can be viewed as a **filtering process of the signal** ***f***

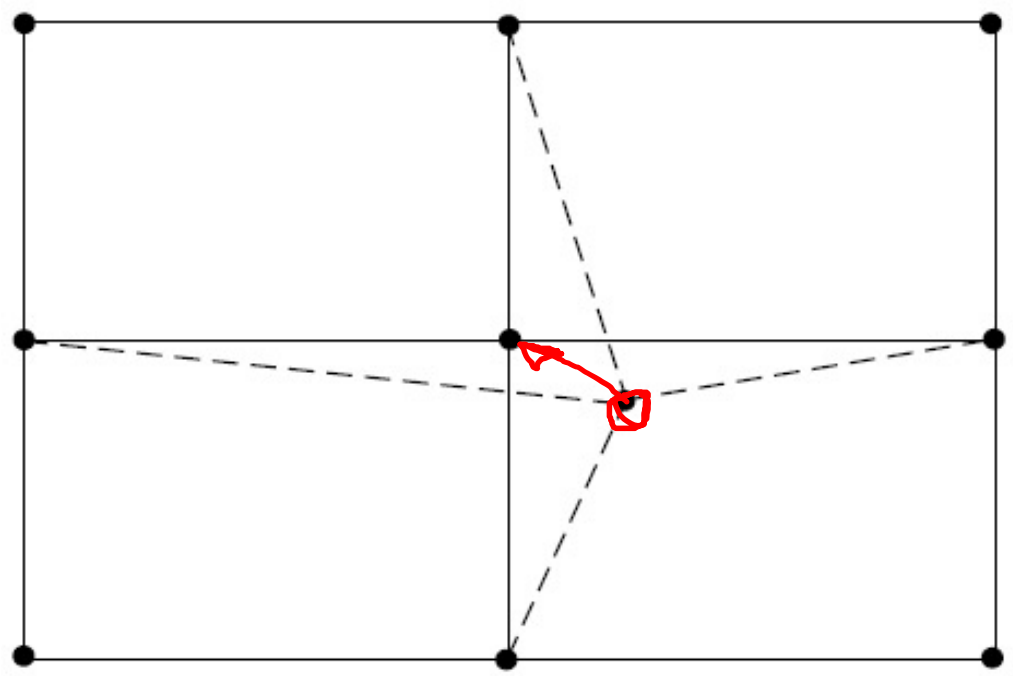
diffusion flow

- widely used to blur images and smooth terrain surfaces
- build a *scale space* describing the evolution of data through time under the blurring/smoothing process



Laplacian Smooth: Explicit

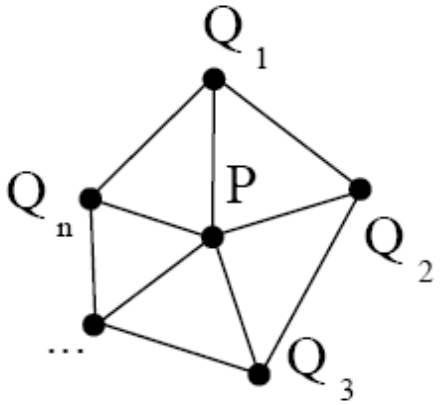
- For each vertex, compute the displacement vector towards the average of its neighbors (Laplacian Operator). Then move each vertex by a fraction of its displacement vector (diffusion over time)



Umbrella Operator

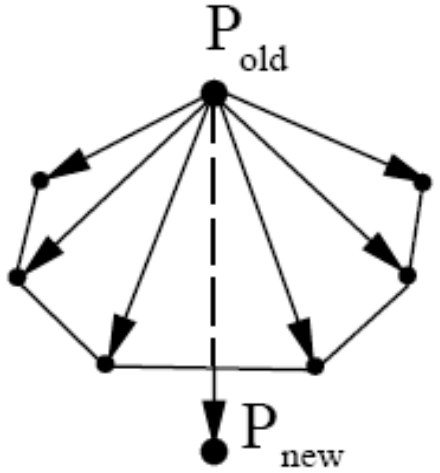
- For each vertex of the mesh:

$$P_{new} = P_{old} + \lambda U(P_{old})$$



- Where **U** is the Umbrella operator

$$U(P) = \frac{1}{\sum_i w_i} \sum_i w_i Q_i - P$$



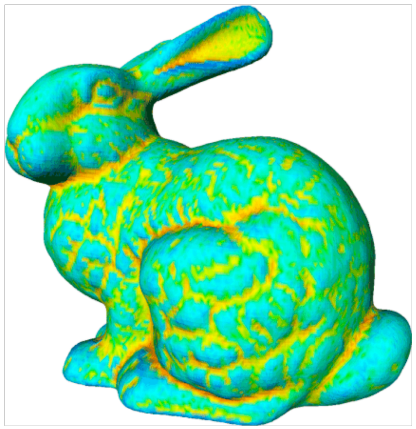
Code

- vcg/complex/algorithms/smooth.h

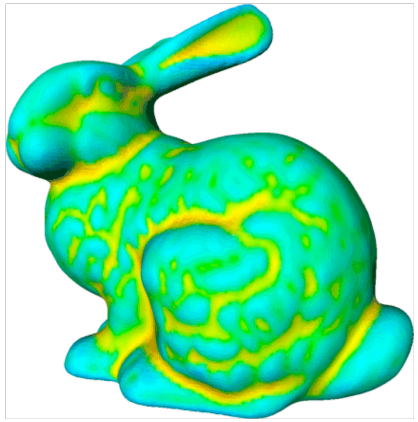
```
static void VertexCoordLaplacianBlend(MeshType &m, int step, float alpha, bool
SmoothSelected=false)
{
    VertexIterator vi;
    LaplacianInfo lpz(CoordType(0,0,0),0);
    assert (alpha<= 1.0);
    SimpleTempData<typename MeshType::VertContainer,LaplacianInfo > TD(m.vert);

    for(int i=0;i<step;++i)
        {
        TD.Init(lpz);
        AccumulateLaplacianInfo(m,TD);
        for(vi=m.vert.begin();vi!=m.vert.end();++vi)
            if(!(*vi).IsD() && TD[*vi].cnt>0 )
            {
                if(!SmoothSelected || (*vi).IsS())
                {
                    CoordType Delta = TD[*vi].sum/TD[*vi].cnt - (*vi).P();
                    (*vi).P() = (*vi).P() + Delta*alpha;
                }
            }
        }
}
```

Example…

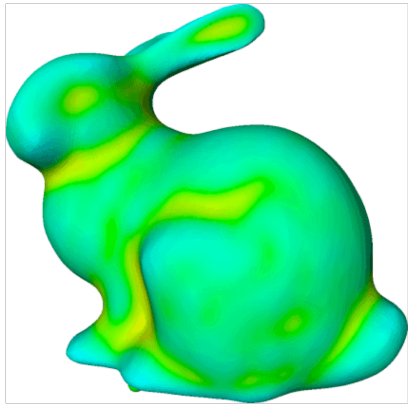
- Shrinking



0 Iterations



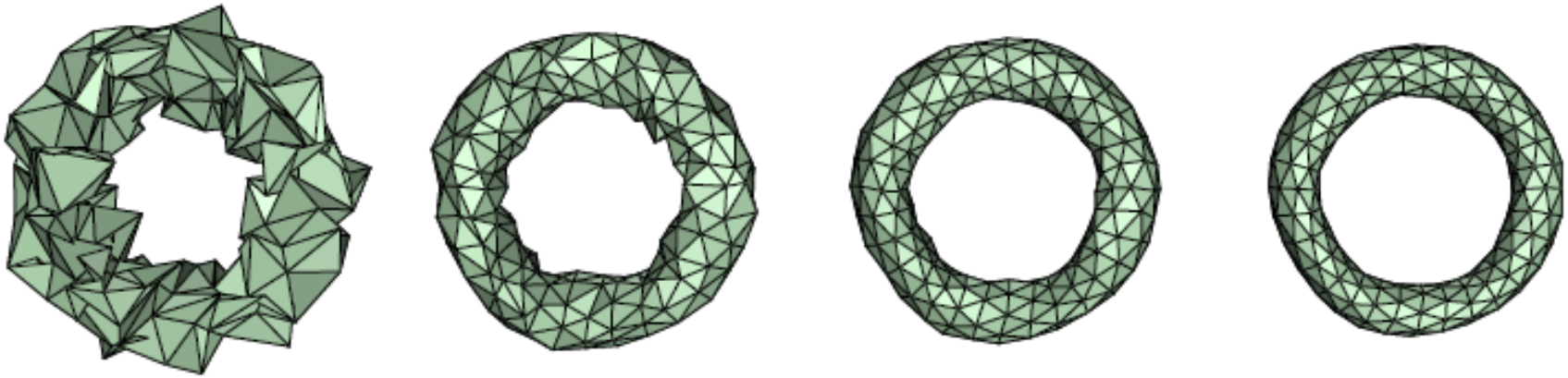
5 Iterations



20 Iterations

Problems…

- Shrinking



Taubin Smoothing

- (2 constants $\lambda>0$ $\mu<0$)

- For each iteration performs 2 steps:
  1. compute the laplacian displacement for each vertex and moves the vertices by λ times this displacement.
  2. Then compute again the laplacian and moves back each vertex by μtimes the displacement.

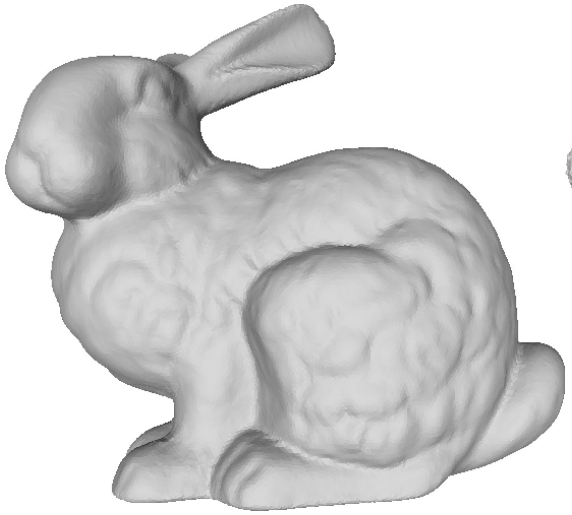
- $\lambda$and μmust be tuned to filter noise at a certain frequency

Code

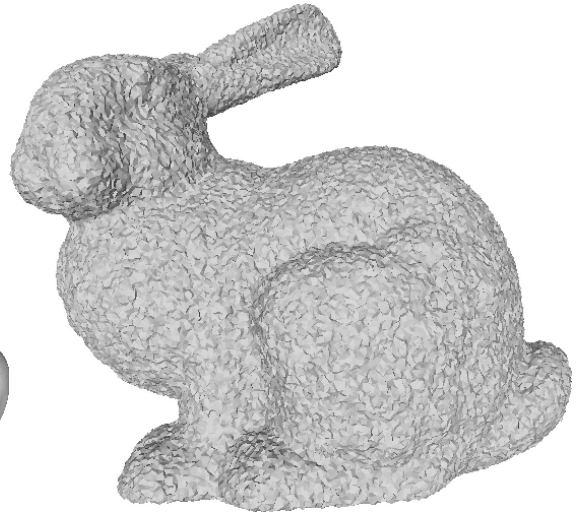
vcg/complex/algorithms/smooth.h

```
static void VertexCoordTaubin(MeshType &m, int step, float lambda, float mu...)
{
    LaplacianInfo lpz(CoordType(0,0,0),0);
    SimpleTempData<typename MeshType::VertContainer,LaplacianInfo > TD(m.vert,lpz);
    VertexIterator vi;
    for(int i=0;i<step;++i)
        {
          if(cb) cb(100*i/step, "Taubin Smoothing");
            TD.Init(lpz);
            AccumulateLaplacianInfo(m,TD);
            for(vi=m.vert.begin();vi!=m.vert.end();++vi)
                if(!(*vi).IsD() && TD[*vi].cnt>0 )
                {
                    if(!SmoothSelected || (*vi).IsS())
                    {
                        CoordType Delta = TD[*vi].sum/TD[*vi].cnt - (*vi).P();
                        (*vi).P() = (*vi).P() + Delta*lambda ;
                    }
                }
            TD.Init(lpz);
            AccumulateLaplacianInfo(m,TD);
            for(vi=m.vert.begin();vi!=m.vert.end();++vi)
                    if(!(*vi).IsD() && TD[*vi].cnt>0 )
                    {
                        if(!SmoothSelected || (*vi).IsS())
                        {
                            CoordType Delta = TD[*vi].sum/TD[*vi].cnt - (*vi).P();
                            (*vi).P() = (*vi).P() + Delta*mu ;
                        }
                    }
            } // end for step
}
```

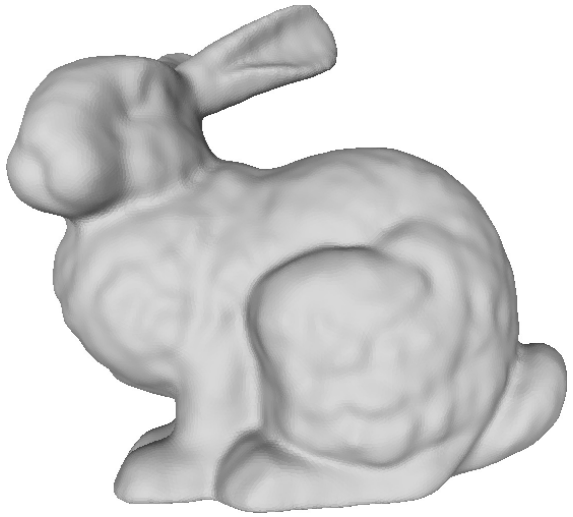
Results



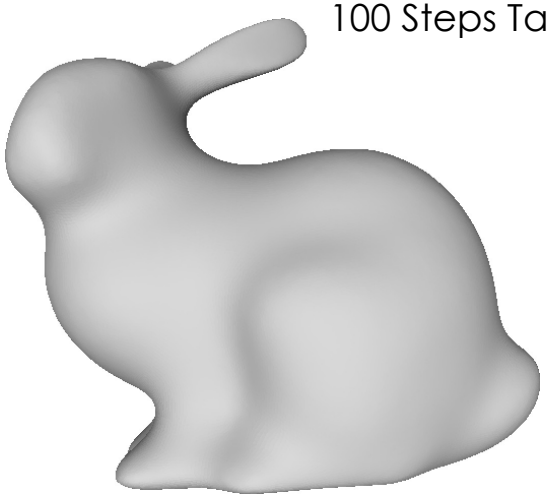
Original



Noise Added



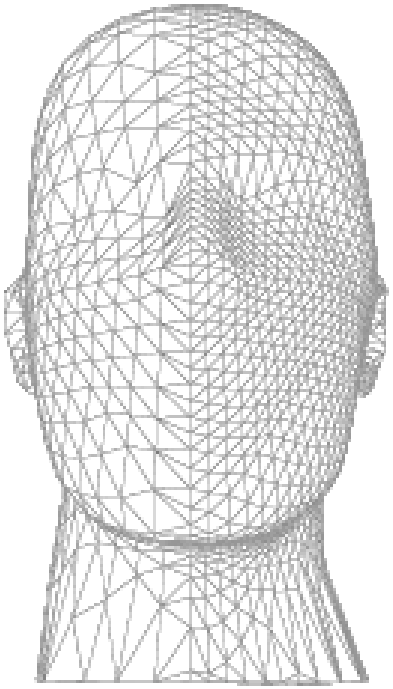
100 Steps Taubin



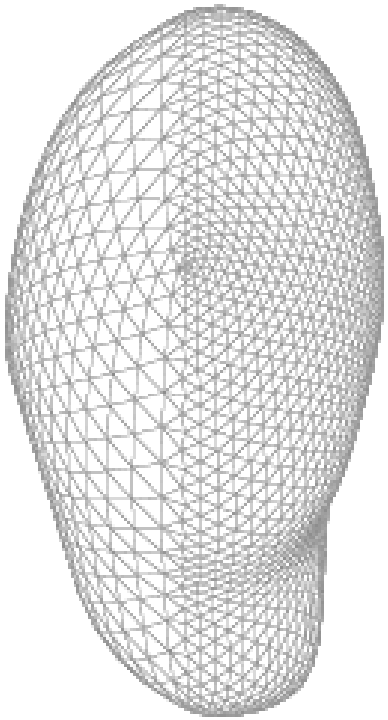
100 Steps Laplacian

Issues: Dependence on Tassellation

- Denser tassellations.. Needs more iterations to converge
- Left part converge faster



Original



Laplacian

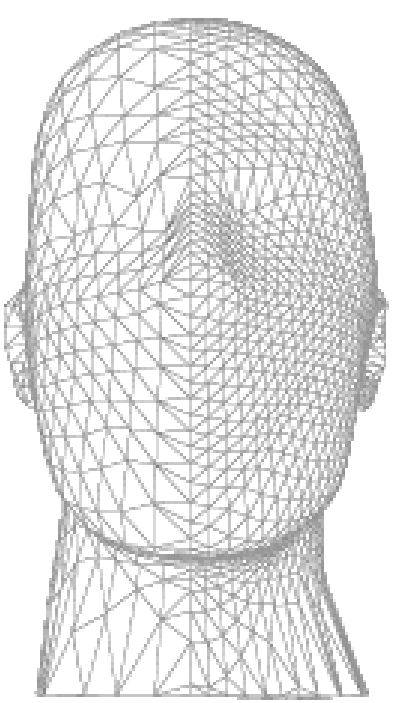
Issues: Dependence on Tassellation

- Substitute Laplacian Operator (that average vertices)
- Une another operator that weiths vertices by considering involved edges

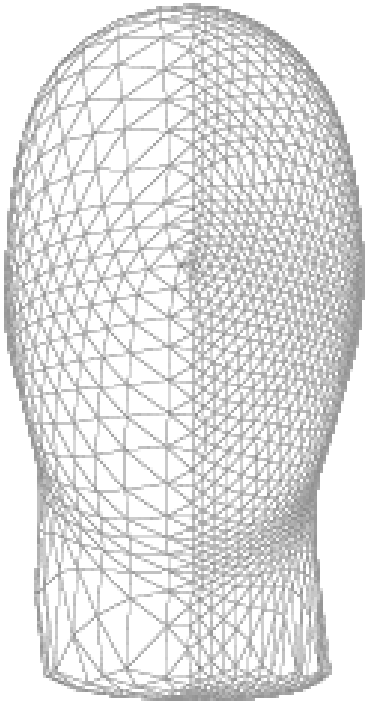
$$U(P) = \frac{2}{E} \sum_i \frac{Q_i}{|e_{ij}|} - P$$

- Where

$$E = \sum_i e_{ij}$$



Original



Scala Dependent

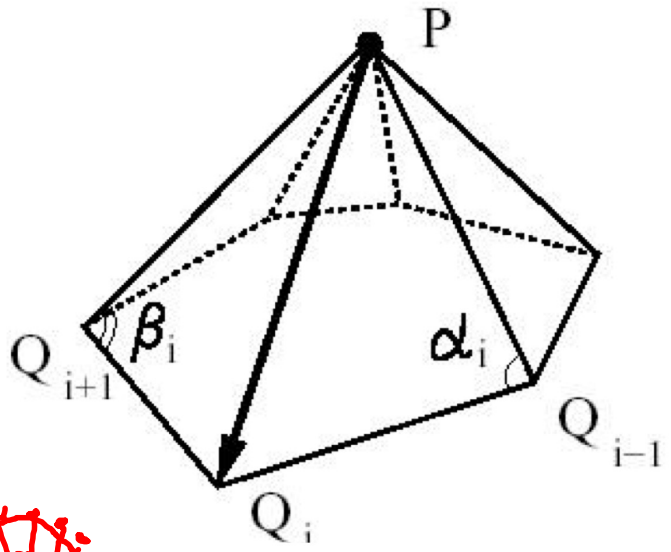
Mean Curvature Flow

- Weight differently considering the different mean curvature
- Use mean curvature in the differental form

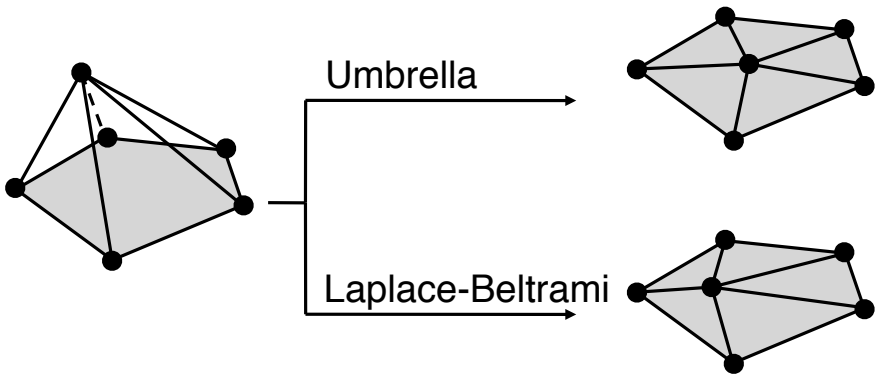
$$P_{new} = P_{old} + \lambda Hn(P_{old})$$

- Laplace beltrami-operator

$$Hn(P) = \frac{1}{4A} \sum_i (\cot \alpha_i + \cot \beta_i)(Q_i - P)$$

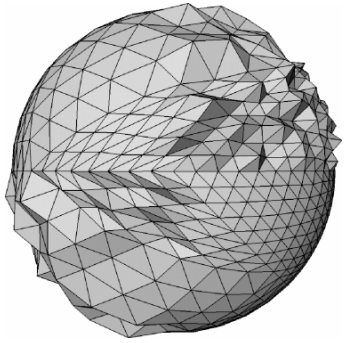


Mean Curvature Flow

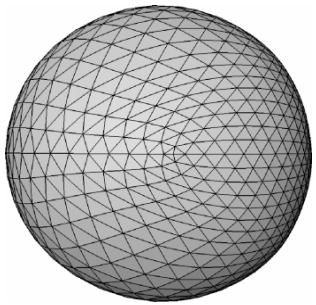


Tangential Drift

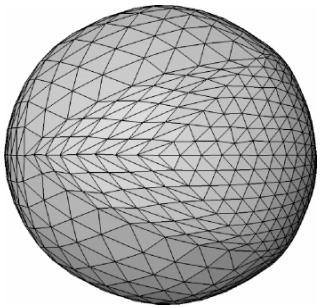
Moves vertices along normal



Original

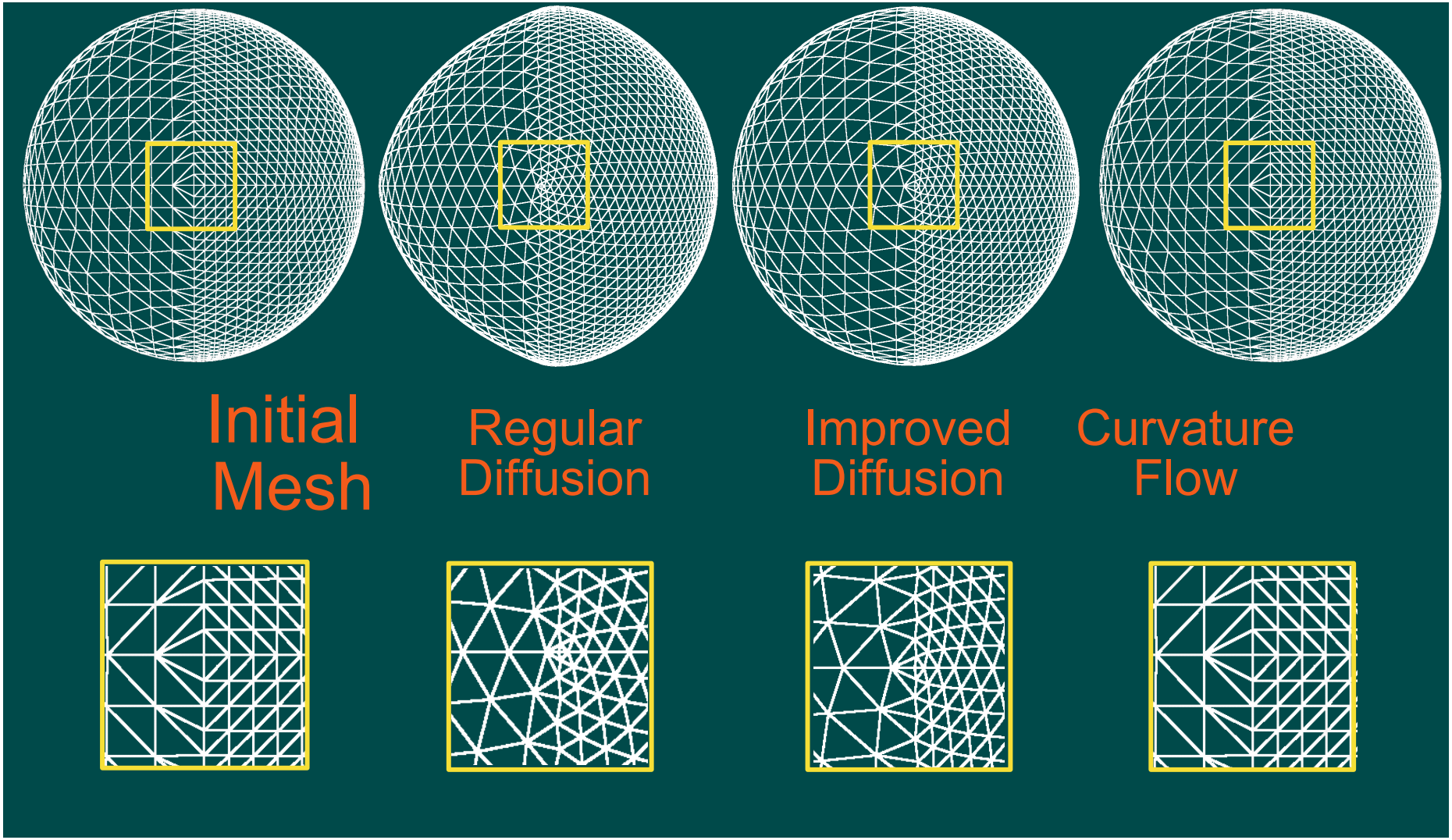


Umbrella



Laplace-Beltrami

Mean Curvature Flow



Smoothing.. Continuous setting

- Minimize the energy dell'energia on a membrane surface
- Find **the minimal area surface**
- The gradient of the energy is the Laplacian (considering a parametrization of M on uv) is given by second partial derivatives…

$$L(M) = \frac{\partial^2(M)}{\partial u^2} + \frac{\partial^2(M)}{\partial v^2}$$

- Discrete of Laplacian (Umbrella operator)

$$L(M) = \frac{1}{n}\sum_j (Q_j - P)$$

Implicit explicit

- **Explicit Euler integration**: resolve the system by iterative substitution for small time step h

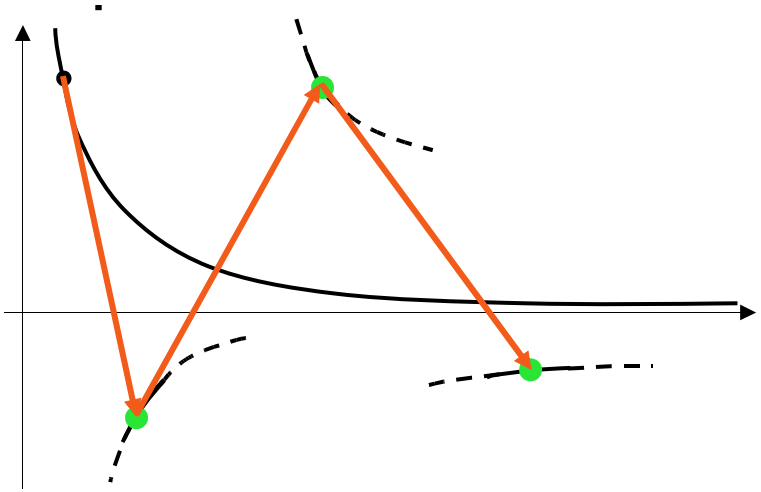
$$\mathbf{f}(t+h) = \mathbf{f}(t) + h\lambda\mathbf{L}\mathbf{f}(t)$$

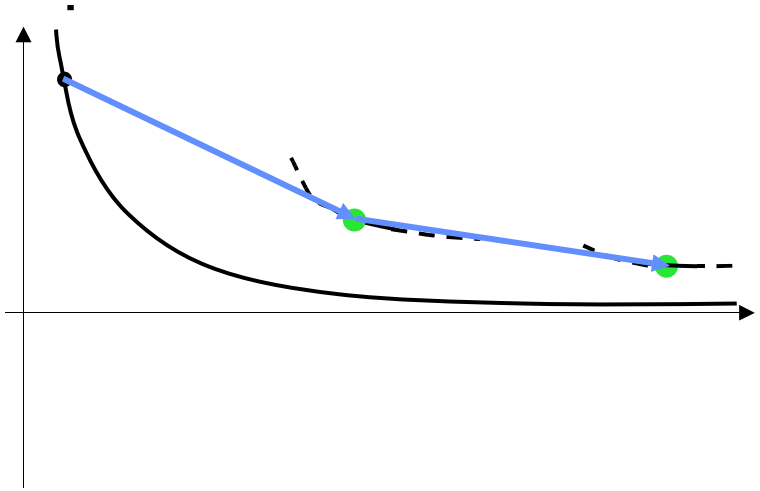
- **Implicit Euler integration**: resolve the following linear system

$$(\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

- the system is very large but sparse

Implicit explicit





explicit	implicit
Easy to code	More difficoult to code
Easy to tune	More difficoult to tune
Need several iter to converge	Instant convergence
Need to tune time step	Stable!

Implicit in VCG

vcg/complex/algorithms/implicit_smooth.h

```
struct Parameter
    {
        //the amount of smoothness, useful only if we set the mass matrix
        ScalarType lambda;
        //the use of mass matrix to keep the mesh close to its original position
        //(weighted per area distributed on vertices)
        bool useMassMatrix;
        //this bool is used to fix the border vertices of the mesh or not
        bool fixBorder;
        //this bool is used to set if cotangent weight is used, this flag to false means uniform laplacian
        bool useCotWeight;
        ...
        //the degree of laplacian
        int degree;
    }
```

```
static void Compute(MeshType &mesh, Parameter &SParam)
```

Smoothing

Geometry process & VCGLib course – Day 4

Nico Pietroni (nico.pietroni@isti.cnr.it)

ISTI – CNR