Perturbing Mesh Vertices with Additive Gaussian Noise

Release 0.00

Davis M. Vigneault[1,2,3]

November 25, 2016

[1]Institute of Biomedical Engineering, Department of Engineering Science, University of Oxford, Oxford, United Kingdom.
[2]Department of Radiology and Imaging Sciences, Clinical Center, National Institutes of Health, Bethesda, MD, United States.
[3]Sackler School of Graduate Biomedical Sciences, Tufts University School of Medicine, Boston, MA, United States.

Abstract

This brief document provides an introduction to an external ITK module, `DVMeshNoise`. This module provides classes for perturbing the positions of the vertices of either an `itk::Mesh` or `itk::QuadEdgeMesh` with Gaussian noise. This may be useful in testing the robustness of an algorithm to small changes in the input mesh, augmenting datasets for machine learning, and counteracting deleterious effects which highly regular regions of a mesh may occasionally have on mesh processing. After demonstrating basic usage, we present a practical example to show how these classes can be incorporated into a pipeline to improve mesh quality.

Contents

1 Interface and Basic Usage

To illustrate basic usage, we generate a sphere, perturb the vertices with Gaussian noise, and write the "clean" and "noisy" meshes to file. We begin with the necessary `includes` and `typedefs`.

```
#include <itkMesh.h>
#include <itkRegularSphereMeshSource.h>
#include <itkAdditiveGaussianNoiseMeshFilter.h>
#include <itkMeshFileWriter.h>

typedef double     TPixel;
const unsigned int Dimension = 3;

typedef itk::Mesh< TPixel, Dimension > TMesh;
typedef itk::RegularSphereMeshSource< TMesh > TSphere;
typedef itk::AdditiveGaussianNoiseMeshFilter< TMesh, TMesh > TNoise;
typedef itk::MeshFileWriter< TMesh > TMeshWriter;
```

We have chosen to illustrate our pipeline using an `itk::Mesh`, as this is expected to the most common use case in the community. However, if we had instead required an `itk::QuadEdgeMesh`, we would have included `itkAdditiveGaussianNoiseQuadEdgeMeshFilter.h` and adjusted our `typedefs` appropriately; all other steps are identical.

Inside `main`, we generate a sphere and write it to file.

```
  const auto sphere = TSphere::New();
  sphere->SetResolution( 2 );

    {
    const auto writer = TMeshWriter::New();
    writer->SetInput( sphere->GetOutput() );
    writer->SetFileName( "../data/sphere.vtk" );
    writer->Update();
    }
```

Next, we add Gaussian noise to the vertices (setting the mean, standard deviation, and seed), and write the perturbed mesh to file.

```
  const auto noise = TNoise::New();
  noise->SetInput( sphere->GetOutput() );
  noise->SetSeed( 0 );
  noise->SetMean( 0.0 );
  noise->SetSigma( 0.05 );

    {
    const auto writer = TMeshWriter::New();
    writer->SetInput( noise->GetOutput() );
```

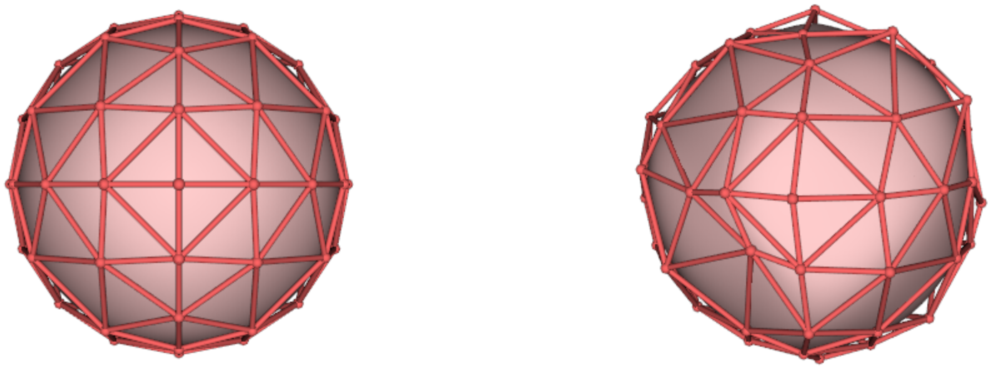

Figure 1: A sphere before (left) and after (right) perturbation with Gaussian noise. We have visualized the meshes by displaying the wireframe (red) as well as the underlying Loop subdivision surface (pink). The source code used to generate this visualization may be found in `dv-wireframe.cxx` and included files.

```
    writer->SetFileName( "../data/sphere_noise.vtk" );
    writer->Update();
    }
```

The meshes before and after perturbation are shown in Figure 1.

2 Application to Mesh Decimation

Generating a mesh from a binary image using marching cubes and decimating the result is a common motif in the analysis of volumetric medical images. Perhaps surprisingly, the quality of the resulting mesh can suffer in regions with very high regularity, such as a perfectly flat area where a structure extends beyond the bounds of the volume.

We begin by creating a spherical mesh using `itk::RegularSphereMeshSource` and converting the mesh to a binary image using `itk::TriangleMeshToBinaryImageFilter`. We ensure that the region of the output image only partially overlaps with the sphere, and pad the output with `itk::ConstantPadImageFilter`, such that the binary image contains a portion which is perfectly flat. Finally, we use `itk::BinaryMask3DMeshSource` to convert the binary image to a mesh. The class which generates this mesh is contained in `dvGenerateHalfSphere.h`.

We then use `itk::QuadricDecimationQuadEdgeMeshFilter` to decimate first the direct output of this algorithm, and second the output of this algorithm after perturbation by a small amount of Gaussian noise ($\sigma = 0.01$). Cursory visual inspection suggests that mesh quality is significantly improved in the decimation of the perturbed mesh (Figure 3) compared with the unperturbed mesh (Figure 2). In particular, the broad, flat "bottom" of the sphere is particularly low quality in the original mesh, with very high valency vertices

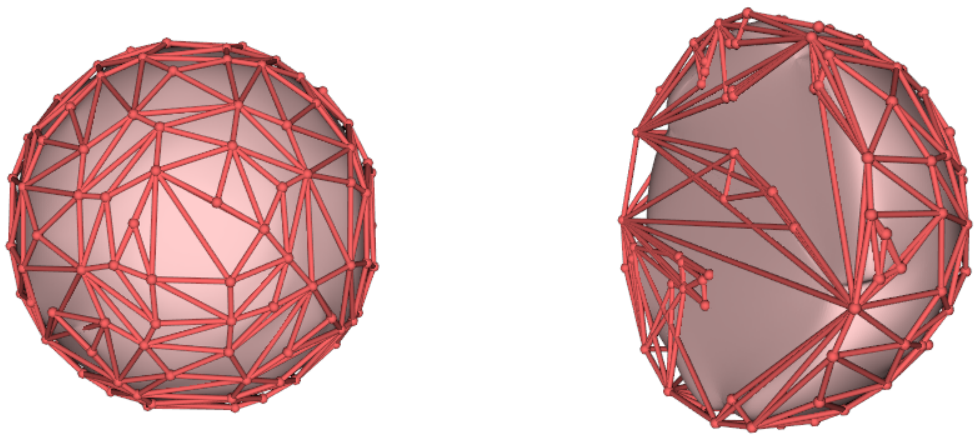

Figure 2: Decimation of the output of a marching cubes mesh generation algorithm results in visually low-quality meshes. We note particular difficulty on the “clean” half of the sphere, with overlapping triangles and areas with very high valency.

and overlapping triangles.

In order to investigate the change in mesh quality, we chose three evaluation metrics: patch area (Figure 4), patch aspect ratio (Figure 5), and valency (Figure 6). For each metric, we plot the histogram and normal probability plot for the unperturbed and perturbed meshes. Histograms provide a generic way of visualizing distributions. Normal probability plots, on the other hand, specifically assess normality by plotting each value against its theoretical quantile for a normally distributed dataset of the same mean and standard deviation; a perfectly normal dataset will lie along a straight line with a coefficient of determination R^2 of one. In the unperturbed case, we find that the histograms have a high standard deviation and significant right skew, with instances of very high triangle area, aspect ratio, and valency. In the perturbed case, we see that the distributions are narrower, with less right skew, indicating greater uniformity. This assessment is confirmed by the normal probability plots, where in each case the coefficient of determination is nearer to unity with perturbation than without.

3 Conclusions

We have presented here two classes, allowing `itk::Mesh` and `itk::QuadEdgeMesh` instances to be perturbed by additive Gaussian noise. Additionally, we have detailed a specific scenario where these classes can be used to quantitatively improve mesh quality by three independent metrics. We hope that the ITK community will find this module useful, and of course welcome feedback and improvements.

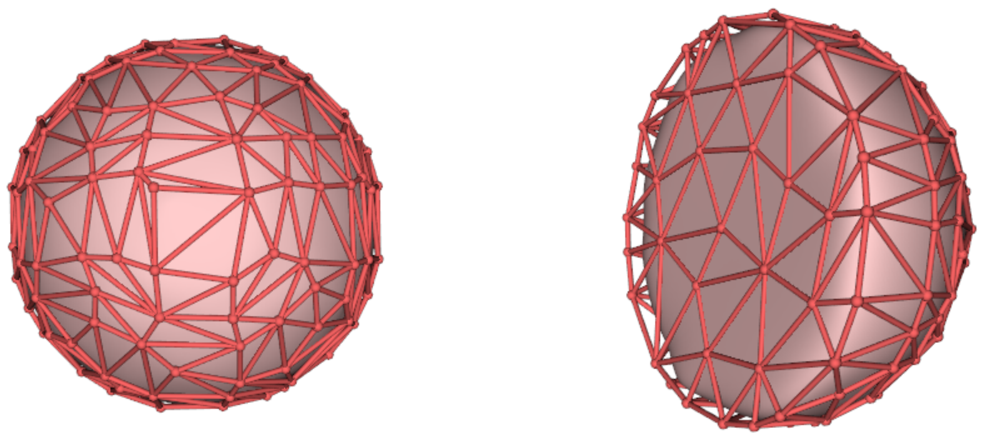

Figure 3: Perturbation with Gaussian noise significantly improves the visual mesh quality of the decimation result compared with the unperturbed mesh, especially in highly regular regions.

Acknowledgments

I'd like to thank my DPhil supervisors, Prof. J. Alison Noble, PhD, and Dr. David A. Bluemke, MD, PhD, for their guidence and support. This work was funded by the NIH-Oxford Scholars Program and the NIH Intramural Research Program.

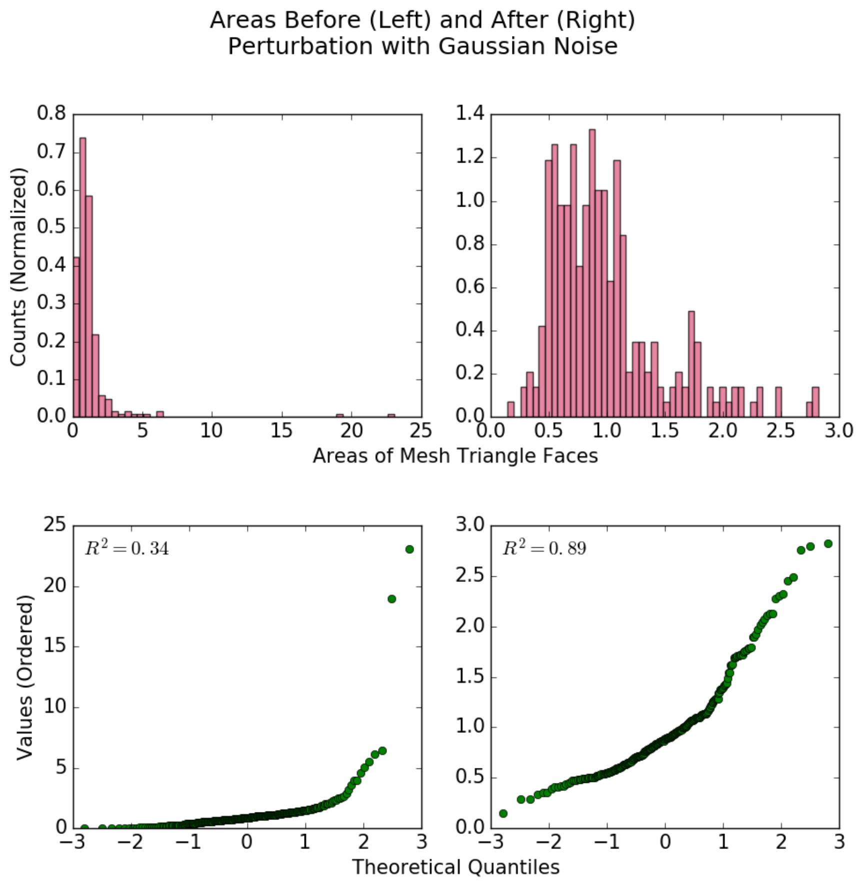

Figure 4: Distribution of patch areas in the clean and noisy meshes.

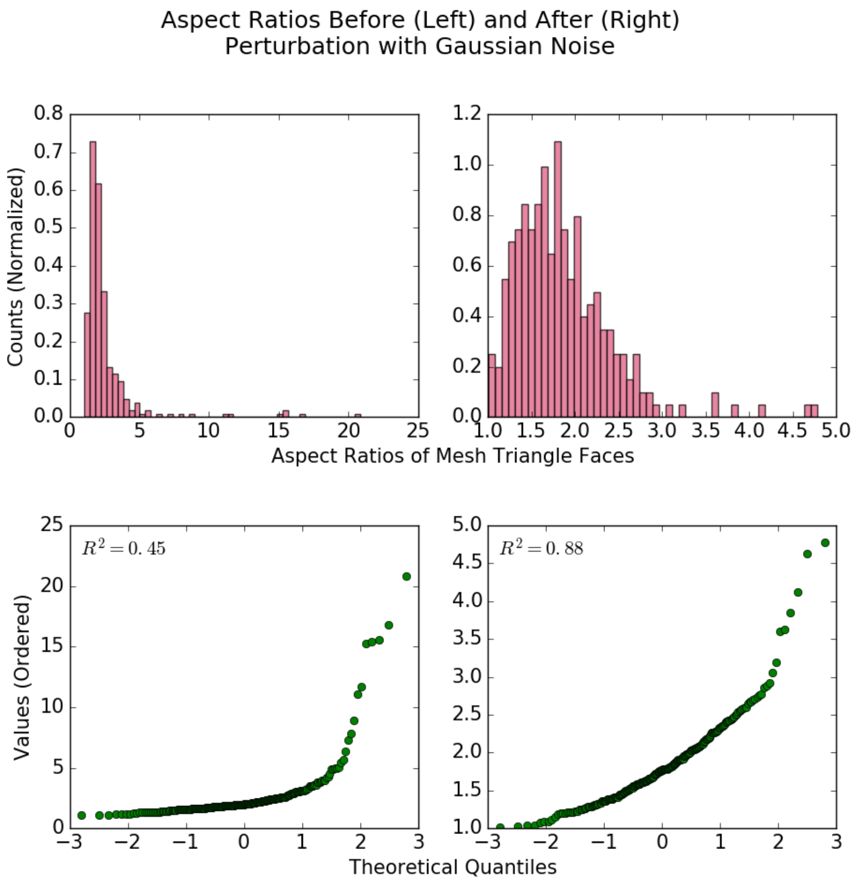

Figure 5: Distribution of patch aspect ratios in the clean and noisy meshes.

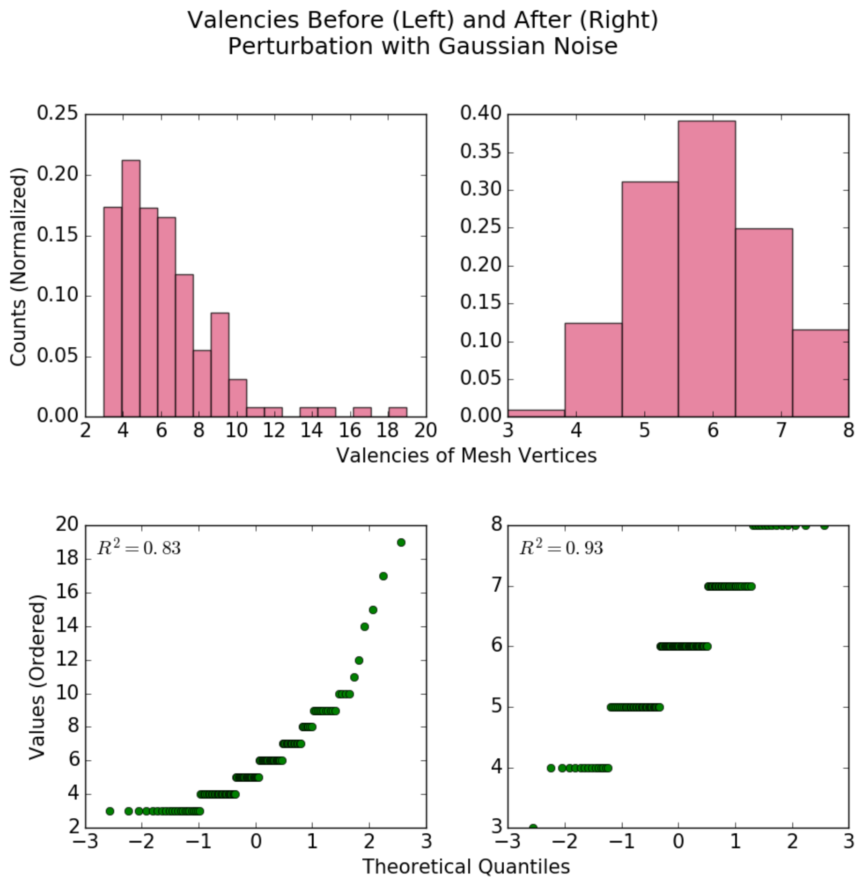

Figure 6: Distribution of valencies in the clean and noisy meshes.