SURFACE TRIANGULATION AND THE DOWNSTREAM EFFECTS ON SURFACE FLATTENING

By

SUDHEER SINGH PARWANA

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

Geometric Modelling Group
Manufacturing and Mechanical Engineering
University of Birmingham
December 2010

SYNOPSIS

Surface triangulation is an active area of research due to its wide usage in a range of different computer aided applications, including computer aided design (CAD), manufacture (CAM) and finite element analysis (FEA). Although these applications are used to create, interrogate, manipulate and analyse surfaces, internally they actually approximate the surface geometry using a triangulation and then operate on the triangles, making them triangulation dependant algorithms. However, despite the reliance on the triangulation by the downstream application, there is very little work which has been focused on the inherent affects of the underlying triangulation on the performance or result of the application. Therefore, the impact of the triangulation on the downstream application is still not well known or defined.

This thesis investigates triangulation and the downstream effects on the triangulation dependant method of surface flattening. Two novel topics are explored, notably right angle triangulation configurations (RATCs) and axis of minimum principal curvature (AMPC) influenced triangulations, each which was found to have an impact on the triangulation dependant method of surface flattening.

Right angle triangles (RATs) are commonly used throughout surface flattening. However, given a set of uniformly sampled points, there are many different ways in which the diagonals can be placed to form a final triangulation consisting of RATs. These different configurations of edges are introduced as RATCs. To investigate the effects of RATCs on surface flattening, three global configurations are proposed; regular, diamond and chevron. In addition, local variations of RATCs are explored by fitting RATCs that best approximated the local Gaussian curvature of the original surface.

Also considered is the influence of triangulation developability and its effects on flattening. Developable surfaces should flatten without inducing any area or shape distortion. However, it is shown that the transfer of this developabilty information from the surface to the triangulation can be lost. It is established that skewing the vertices of a triangulation so that no triangle edges followed along the AMPC, causes the flattening to induce distortion. As the shape of a triangulation is defined by its edges, if no edges follow along the AMPC, this means that the triangulation will not inherit this shape characteristic. Therefore, a new method is proposed to triangulate a surface, whilst ensuring that one edge of every triangle follows along the general direction of the AMPC.

This thesis also introduces a new efficient and robust parametric trimmed surface triangulation method. Efficiency is gained during trimmed curve tracing by minimising the number of cells processed. Key features are the efficient tracing algorithm and knowledge of orientation of the trimming curves is not required. This approach also minimises the occurrence of degenerate triangles and copes with holes independently of the grid size.

ACKNOWLEDGEMENTS

First, I would like to express my gratitude and upmost thanks to my supervisor Dr. Robert Cripps for his guidance, support and friendship throughout my PhD journey. I would also like to thank Prof. Alan Ball and the entire GMG team for their wonderful insight and encouragement whenever I needed inspiration.

Thank you to Delcam International, and in particular the PS-Exchange team, for providing me with valuable industrial work experience and hands-on training in programming.

I am indebted to my parents (Surrinder and Parmod), my sisters (Jyoti and Pooja) and my wife (Harjeet) for all their support and love throughout this PhD and my life.

This research was funded by ESPRC under the Doctoral Training Award programme (EP/P502322/1) and Delcam International.

Finally, I would like to dedicate this work to my beautiful daughter Anaya.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Computer aided engineering (CAE) is now a mature technology that has been widely adopted by industry. CAE suites are a composite of many diverse systems used through a product's development pipeline, including computer aided design (CAD), manufacture (CAM) and finite element analysis (FEA) packages. The use of CAE can aid in reducing lead times (Cader *et al*., 2005) and hence reduce costs in rapid prototyping, testing and production. These factors have contributed to CAE's wide usage in industry, thus encouraging much research and development.

In order to use CAE, a virtual representation of an object is required which can be characterised in a number of different ways. One type of representation uses discrete data (Cripps and Cook, P.R., 1999), where a model is acquired as a set of points (Figure 1.1). These points can then be strategically joined with edges to create a set of triangles that approximates the original shape, known as a triangulation or mesh. Understanding the data provided and knowing how to interpret the points can be a difficult task as it is not guaranteed that the points will be acquired in a logical and structured manner. Furthermore, the points are an approximation of the original geometry; therefore the integrity of the model may be compromised. Triangulation optimisation (also known as remeshing) techniques can be used to modify the density and uniformity of triangles within a triangulation and have been well studied (Shimada and Gossard, 1998; Shu and Boulanger, 2000; Wang et al., 2006). However, these methods do not guarantee that any new points, edges or triangles created will necessarily lie on the original model geometry.

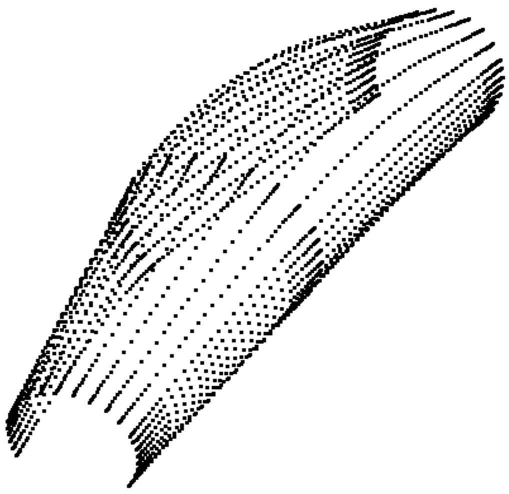


Figure 1.1 Point based model of a BAE systems stylised canopy.

An alternative approach is to use a mathematical representation of a model such as a collection of surfaces (Kumar et *al*., 2001) (Figure 1.2). Again, to use many CAE applications, these mathematical models must be approximated as a set of triangles. In this case the mathematical definitions are evaluated to generate a list of points that are then connected together to form a triangulation (Rockwood *et al*., 1988; Kumar and Manocha, 1995, 1996; Piegl and Tiller, 1998). The advantage here is that the point density, and hence accuracy of the approximation can be controlled with a guarantee that any new points sampled will lie on the actual model (Piegl and Tiller, 1998). Furthermore, the points can be generated in a structured way, making the triangulation process more efficient and robust (Sadoyan *et al*., 2006).

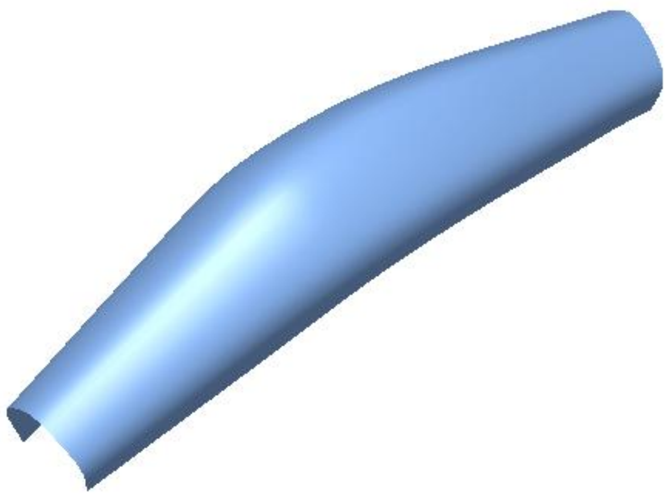


Figure 1.2 Surface model of the same canopy model shown in Figure 1.1.

Surfaces provide a high degree of freedom for modellers when constructing complex geometries (Rogers and Adams, 1990; Piegl and Tiller, 1997). Therefore, this thesis will be concerned with surfaces that are represented as non-uniform rational B-splines (NURBS), more specifically focusing on non-rational forms (NUBS). This representation of surfaces is conventionally adopted throughout industry due to its properties and flexibility in shape manipulation and design (Rogers and Adams, 1990). Another advantage of NUBS form is that they inherently encompass another powerful and widely used representation called Bézier (Piegl and Tiller, 1997) as a subset. Therefore, NUBS curves and surface can be decomposed and converted into Bézier form (Section 1.2.3); a feature that has been used in existing triangulation algorithms (Rockwood *et al*., 1988; Kumar and Manocha, 1995, 1996).

Once a NUBS surface has been triangulated, it can then be used for a wide range of CAE downstream applications such as interrogations, visualisations, computer numerical control (CNC) machine path generation, FEA and flattening. As these methods only have access to the initial surface geometry through its triangulation, they could be characterised as 'triangulation dependant' processes (Parwana and Cripps, 2009).

Despite there being many different algorithms available for generating a triangulation from a surface representation, little attention has been targeted on the downstream effects of the triangulation on different engineering applications. For this reason, choosing a triangulation algorithm, to approximate a surface, comes down to a user's personal preference. Furthermore, the geometric characteristics transferred from the initial surface to the triangulation and the impact this has on a downstream application are not well understood (Parwana and Cripps, 2009).

This thesis will study surface triangulation and investigate the effects it imposes on the performance of surface flattening. Flattening is a triangulation dependant method which is used by industries such as shoe, ceramic decoration and textiles (McCartney et al., 1999; Wang et al., 2002; Cadar et al., 2005). The purpose is to iteratively map a three-dimensional (3D) triangulation into a two-dimensional (2D) plane (Chapter 4). It is known that altering algorithm parameters can cause different results from a flattening simulation (Cadar et al., 2005) (Section 4.2); however this thesis will aim to isolate specific behaviours that are solely attributed to changes in the triangulation.

This thesis will predominantly investigate the effects of right angle triangles (RATs) on flattening. As RATs can be easily generated from a uniformly sampled grid of points they are commonly used throughout research and industry (McCartney et al., 1999; Wang et al., 2002). A new triangulation variable called right angle triangle configurations (RATCs) is introduced (Parwana and Cripps, 2009), which refers to the connectivity of RATs within a triangulation, and its impact on flattening is investigated. In addition, this thesis will investigate and establish specific surface characteristics that can have an influence on the flattening result. These characteristics will then be incorporated into the final triangulation to better reflect the surface geometry and flattening simulations.

The remainder of this chapter will provide details of the NUBS definition of curves and surfaces, as well as their characteristics. A brief review of some differential geometry elements will also be provided. It will then present details of how a triangulation topology is represented, for computational purposes, along with some triangle mathematics which will aid in the understanding of this thesis. The chapter will conclude with an overview of the thesis structure by outlining the contents of each chapter.

1.1 NUBS Curves and Surfaces

NUBS curves and surfaces are used widely among commercial CAD software systems such as Dessault’s Catia and Delcam’s PowerShape. Since they are parametrically defined polynomial entities, they are well suited for software implantation (Lockyer, 2007).

The B-Spline basis was originally developed for statistical data smoothing by Schoenburg (1946) before Cox (1972) and de Boor (1972) both independently presented a more numerically stable recursive definition that allowed the basis functions to be established more efficiently. These B-Spline basis were then introduced to CAGD in the context of parametrically defined curves by Riesenfeld (1973) and later extended to NURBS form by Versprille (1975).

1.1.1 B-Spline Basis Functions

The B-Spline basis functions can be calculated using the recurrence formula (Cox, 1972; de Boor, 1972, 2001) since it is the most useful for computer implementation.

The *p*th degree (order *p*+1) B-Spline basis function is calculated as a function of a non-decreasing sequence of real numbers, u_i, called knots defined by the knot vector **U**, defined as

$$\mathbf{U} = \{u_0, \ldots, u_{r-1}\} \qquad u_i \le u_{i+1}, \;\; 0 \le i \le r-2 \tag{1.1}$$

The number of knots is *r*.

The *i*th normalised B-Spline basis function, $N_{i,p}(u)$, is defined as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{1.2a}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{1.2b}$$

where the quotient 0/0 = 0.

The *k*th derivative of the $N_{i,p}(u)$ is denoted as $N_{i,p}^{(k)}(u)$ and is given by (Piegl and Tiller (1997)),

$$N_{i,p}^{(k)}(u) = p\left(\frac{N_{i,p-1}^{(k-1)}(u)}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}^{(k-1)}(u)}{u_{i+p+1} - u_{i+1}}\right) \qquad \begin{array}{l} 0 \le i \le r-1 \\ 0 \le k \le p-1 \end{array} \tag{1.3}$$

If the knot vector is of the form

$$\mathbf{U} = \{\underbrace{0,\ldots,0}_{p+1},\underbrace{1,\ldots,1}_{p+1}\} \tag{1.4}$$

the normalised B-Spline basis functions are equivalent to the Bernstein polynomials (Piegl and Tiller, 1997), which are used as the basis functions for the Bézier form. Knot vectors of the form (1.4), i.e. the end knots have *multiplicity* p+1 with no repeated interior knots, are called open clamped knot vectors.

1.1.2 B-Spline Curve

A pth degree NURBS curve is defined by

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \qquad \begin{array}{l} w_i > 0 \\ u_p \le u \le u_{r-p-1} \end{array} \tag{1.5}$$

where $\mathbf{P}_i$ are the 2D or 3D control points, w_i are the weights and $N_{i,p}(u)$ are the pth degree B-Spline basis functions defined on the non-uniform open clamped knot vector

$$\mathbf{U} = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{r-p-2}, \underbrace{b,\ldots,b}_{p+1}\} \tag{1.6}$$

For a single curve span, the number of control points n is always equal to the order of the curve and the number of knots $r = n + p + 1$ (Figure 1.3).

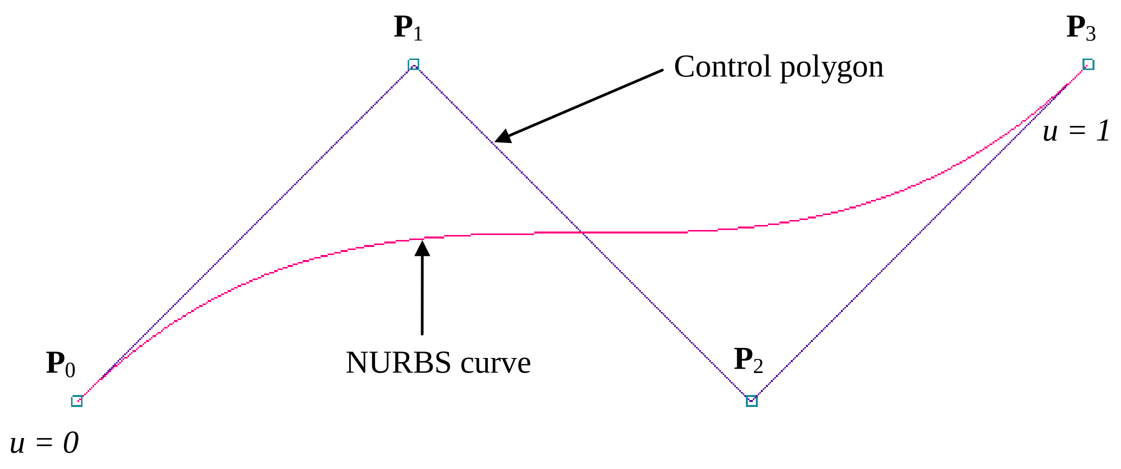
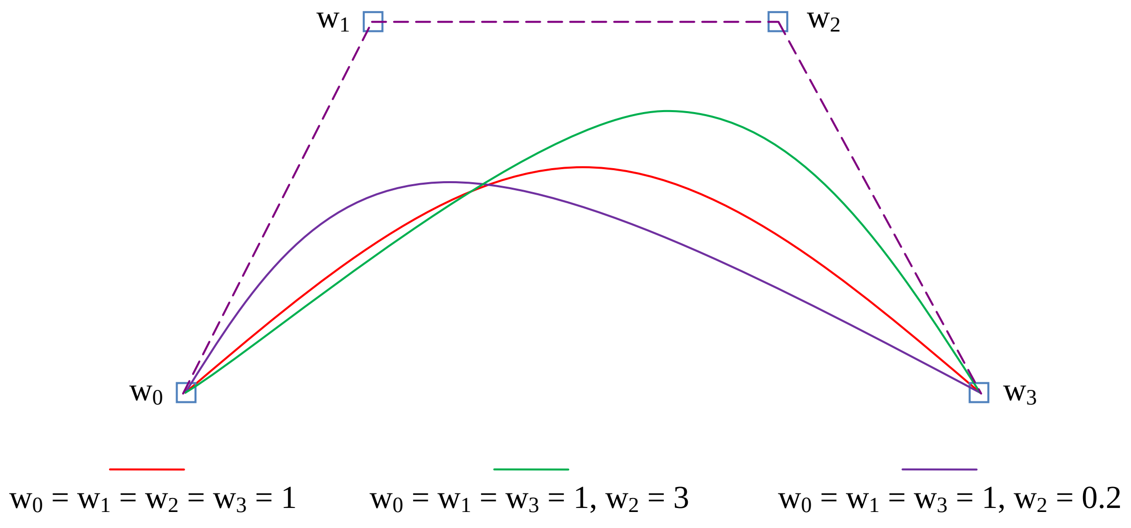


Figure 1.3 An example of a degree 3 NURBS curve.

The weights are applied to the individual control points and act as magnets, where a control point, $\mathbf{P}_i$, with a weight value greater (less) than the other control points will attract (repel) the curve towards (away from) $\mathbf{P}_i$ (Figure 1.4). Ensuring that $w_i > 0$ for $0 \le i < n$ will produce a curve that will be completely contained within the convex hull of the control points (O'Rourke, 2001). Furthermore, if $w_i = c \forall i$ where $c > 0$, there is no bias on the curve and (1.5) is reduced to non-rational form (NUBS)

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i \qquad u_p \le u \le u_{r-p-1} \tag{1.7}$$

The basis functions are defined on the knot vector shown in (1.6).



w_1
w_2
w_0
w_3
$w_0 = w_1 = w_2 = w_3 = 1$
$w_0 = w_1 = w_3 = 1,\ w_2 = 3$
$w_0 = w_1 = w_3 = 1,\ w_2 = 0.2$

Figure 1.4 Degree 3 NURBS curve and the changes applied to the shape by altering the magnitude of the weight values.

The NUBS curve can be differentiated with respects to the parameter *u*. The first and second derivatives, $\mathbf{C}_u(u)$ and $\mathbf{C}_{uu}(u)$ are given by

$$\mathbf{C}_u(u) = \sum_{i=0}^{n} N_{i,p}^{(1)}(u)\mathbf{P}_i \qquad u_p \le u \le u_{r-p-1} \tag{1.8}$$

$$\mathbf{C}_{uu}(u) = \sum_{i=0}^{n} N_{i,p}^{(2)}(u)\mathbf{P}_i \qquad u_p \le u \le u_{r-p-1} \tag{1.9}$$

1.1.3 B-Spline Surface

A NURBS surface is a bi-directional extension of the curve form, where a surface of degree *p* in the *u* direction and degree *q* in the *v* direction is defined by

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad \begin{array}{l} u_p \le u \le u_{r-p-1} \\ v_q \le v \le v_{s-q-1} \\ w_{i,j} > 0 \end{array} \tag{1.10}$$

where $\mathbf{P}_{i,j}$ is the control polygon net, $w_{i,j}$ weights, and $N_{i,p}(u)$ and $N_{j,q}(v)$ are the normalised B-Spline basis functions defined on the clamped knot vectors

$$\mathbf{U} = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{r-p-2},\underbrace{b,\ldots,b}_{p+1}\} \tag{1.11}$$

$$\mathbf{V} = \{\underbrace{c,\ldots,c}_{q+1}, v_{q+1},\ldots,v_{s-q-2},\underbrace{d,\ldots,d}_{q+1}\} \tag{1.12}$$

The number of control points in the *u* and *v* directions is always $n = p+1$ and $m = q+1$ respectively and the number of knots in the *u* and *v* directions is $r = n + p + 1$ and $s = m + q + 1$ respectively. Figure 1.5, shows an example of a NURBS surface and illustrates how the 2D parameter domain is mapped into 3D object space to form the surface.

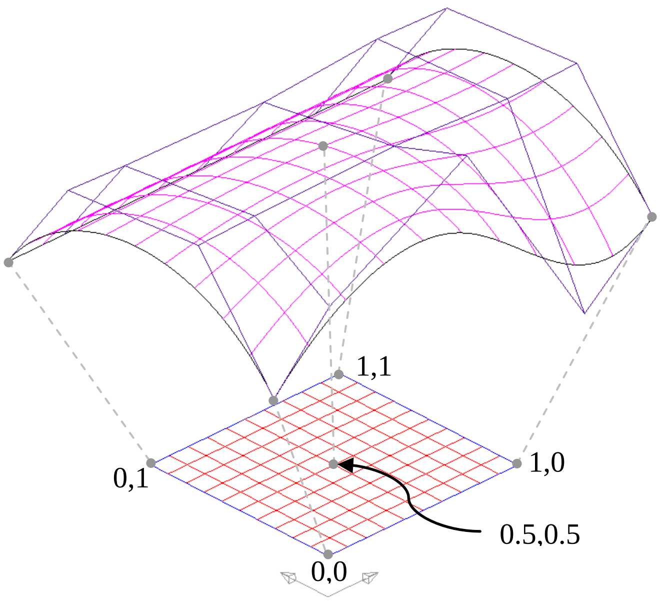


Figure 1.5 Shows a surface defined on the parameter plane, where $u,v \in [0,1]$, and the mapping from 2D to 3D object space to form the surface.

The non-rational definition of the surface (NUBS) is given by $w_{i,j} = c \forall i, j$ resulting in

$$\mathbf{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \mathbf{P}_{i,j} \qquad u_p \le u \le u_{r-p-1}, \;\; v_q \le v \le v_{s-q-1} \tag{1.13}$$

The basis functions are defined on the knot vectors shown in (1.11) and (1.12).

B-Spline surfaces can be differentiated with respects to both *u* and *v*. The first derivatives, $\mathbf{S}_u(u, v)$ and $\mathbf{S}_v(u, v)$ are given by

$$\mathbf{S}_u(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}^{(1)}(u) N_{j,q}(v) \mathbf{P}_{i,j}, \qquad u_p \le u \le u_{r-p-1}, \;\; v_q \le v \le v_{s-q-1} \tag{1.14}$$

$$\mathbf{S}_v(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}^{(1)}(v) \mathbf{P}_{i,j}, \qquad u_p \le u \le u_{r-p-1}, \;\; v_q \le v \le v_{s-q-1} \tag{1.15}$$

The mixed partial derivative $\mathbf{S}_{uv}(u, v)$ is given by

$$\mathbf{S}_{uv}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}^{(1)}(u) N_{j,q}^{(1)}(v) \mathbf{P}_{i,j}, \qquad u_p \le u \le u_{r-p-1}, \;\; v_q \le v \le v_{s-q-1} \tag{1.16}$$

The second derivatives, $\mathbf{S}_{uu}(u, v)$ and $\mathbf{S}_{vv}(u, v)$ are given by

$$\mathbf{S}_{uu}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}^{(2)}(u) N_{j,q}(v) \mathbf{P}_{i,j}\,, \quad u_p \le u \le u_{r-p-1}, \quad v_q \le v \le v_{s-q-1} \tag{1.17}$$

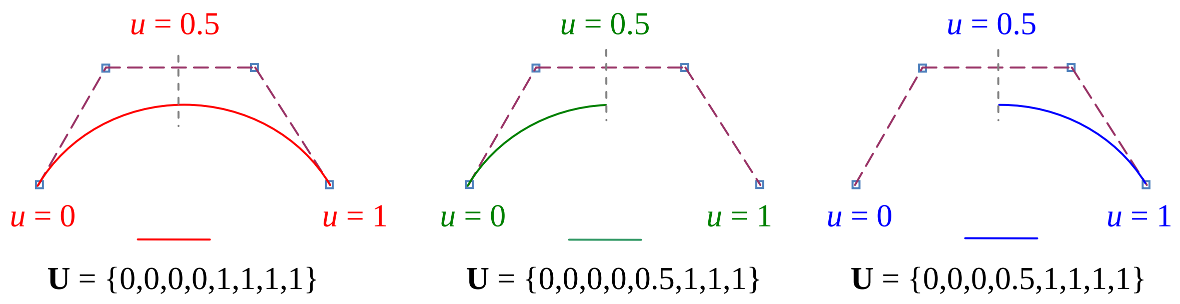
$$\mathbf{S}_{vv}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}^{(2)}(v) \mathbf{P}_{i,j}\,, \quad u_p \le u \le u_{r-p-1}, \quad v_q \le v \le v_{s-q-1} \tag{1.18}$$

1.2 Knot Vectors

The knot vectors define the parameter domain of the NUBS and can be used to convert NUBS into Bézier. The following properties of knots are explained in the context of NUBS curves but equally apply to surfaces.

1.2.1 Knots and the Parameter Domain

Knots u_0 and u_{r-1} are the minimum and maximum limits of the parameter domain respectively. The NUBS curve is defined between the parameter values determined by knots u_{p+1} and u_{r-p-1} (Figure 1.6).



u = 0.5
u = 0
u = 1
U = {0,0,0,0,1,1,1,1}
u = 0.5
u = 0
u = 1
U = {0,0,0,0,0.5,1,1,1}
u = 0.5
u = 0
u = 1
U = {0,0,0,0.5,1,1,1,1}

Figure 1.6 Shows the different cubic NUBS curves generated by altering the knot

A knot vector with end knots repeated p+1 times ensures that the curve starts at the first control point, since $u_o = u_{p+1}$, and ends at the final control point, since $u_{r-1} = u_{r-p-1}$. This type of knot vector is called open clamped. Unless otherwise stated, the remainder of this thesis assumes all knot vectors to be open clamped between 0 and 1.

1.2.2 Knot Spans

Since the knot vector is intrinsically connected to the parameterisation of the curve, the number of knot spans, where a knot span is present when $u_i \neq u_{i+1}$, $0 \leq i \leq r-2$, equals the number of curve spans (Figure 1.7). Therefore, in general, assuming that a knot vector (with *r* knots) has no repeating interior knots, the number of *p*th degree curve spans *cs* is given by

$$cs = r - 2p - 1 \tag{1.19}$$

Having repeated interior knots has the effect of reducing the degree continuity between the curve spans (Piegl and Tiller (1997)).

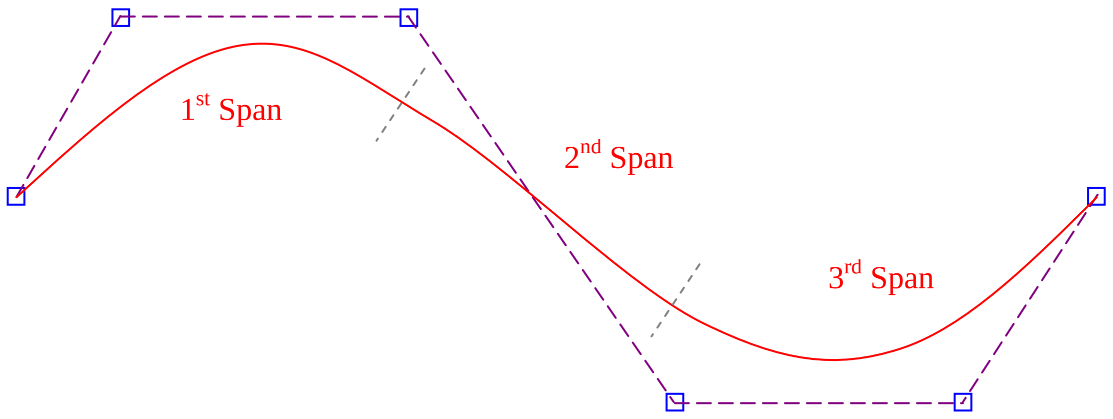


Figure 1.7 A degree 3 NUBS curve with three curve spans.

1.2.3 Knot Insertion

A *p*th degree NUBS curve, consisting of *cs* spans, can be subdivided using knot insertion and involves strategically inserting knots into the current knot vector, then adding and adjusting control points to yield a new description for the same curve. By inserting multiple knots so that each interior knot has multiplicity *p* will generate enough control points to represent each curve span as a *p*th degree Bézier segment (Figure 1.8).

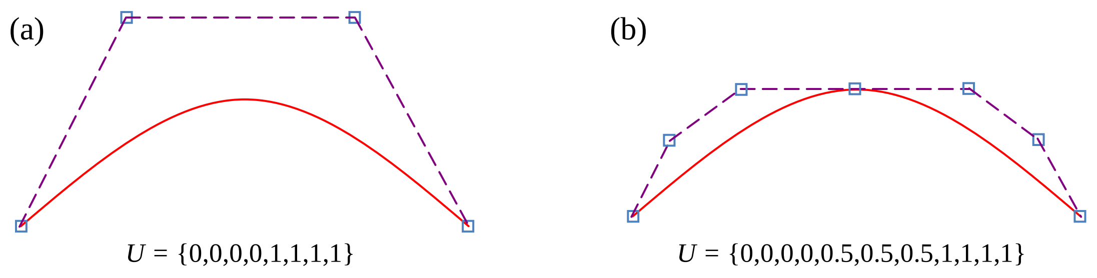


Figure 1.8 (a) Shows an degree 3 open clamped NUBS curve. (b) Shows the same curve after knot insertion.

The modification of control points is based on de Casteljau's algorithm, which is a Bézier subdividing technique. Knot insertion is not directly used in this thesis, therefore the details have been omitted. Full details can be found in Piegl and Tiller (1997).

1.3 Subdivision of NUBS

Many trimmed surface triangulation algorithms require the subdivision of the initial surface into rectangular cells and the subdivision of curves into linear segments (Figure 1.9).

There are two general methods of subdivision: uniform and geometric. This subsection explains the basic concepts and differences between the two methods in the context of curves, however they can be extended bi-directionally to apply to surfaces.

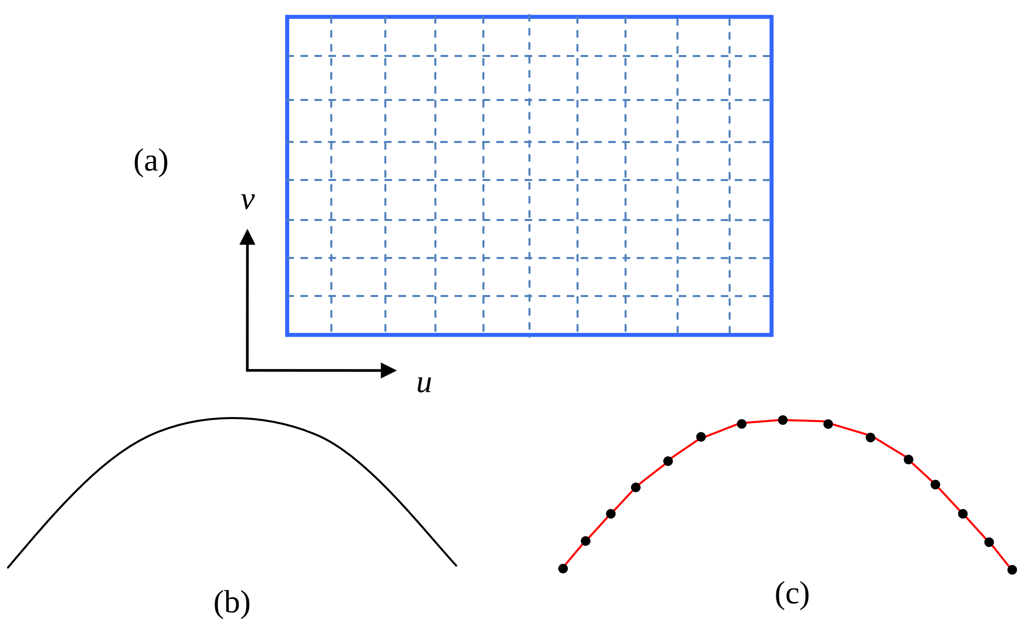


Figure 1.9 (a) Uniformly subdivided surface parameter domain. (b) Curve and (c) its approximation using linear segments.

1.3.1 Uniform Subdivision

Uniform subdivision of a curve is where the parameter domain is split at equal intervals. The parameter increment is a function of the number of steps, which is calculated in real space. It is calculated in this manner so that the distance between consecutive points on the curve in real space does not exceed a user specified tolerance.

Therefore, if a NUBS curve is defined within the parameter domain [$u_{\min}$, $u_{\max}$], and the number of steps that the curve should be split into is $step_u$, then the parameter increment, $u_{increment}$ for the subdivision is defined as

$$u_{incerment} = \frac{u_{\max} - u_{\min}}{step_u} \tag{1.20}$$

Despite its simplicity, uniform subdivision can be problematic for NUBS curves. This is because a single NUBS curve can be a piecewise representation of multiple curves over a single parameter domain (Section1.2.2); however, this does not guarantee that the parameterisation will be distributed evenly across all the piecewise curves.

For example, the NUBS curve represented in Figure 1.10 consists of two curve spans. The curve has been subdivided into 20 steps with an equal parameter increment. As can be seen the 10 points on the first span are more tightly packed than the second span. If the user required that the sizes of each linear segment be the same (within an acceptable tolerance) this would not always be possible using uniform subdivision and the NUBS representation directly.

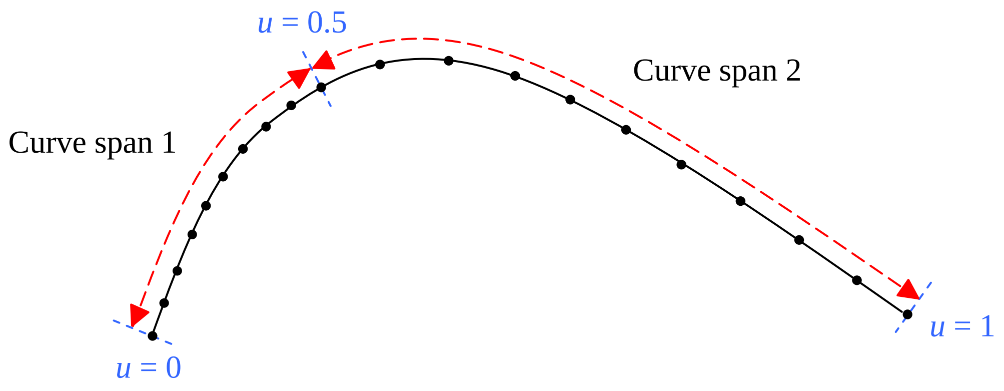


Figure 1.10 NUBS curve with two spans and unevenly distributed parameterisation.

To rectify this problem, each NUBS curve span, between the knot interval $[u_i, u_{i+1}]$, can be subdivided using a different parameter increment. This will result in a final subdivision where the lengths of the linear segments approximating the curves are the same (within a tolerance).

1.3.2 Bézier Uniform Subdivision

An alternative approach for improving the uniform subdivision of a NUBS curve is to convert each curve span into a Bézier curve segment, using knot insertion (Section 1.2.3). A Bézier curve represents a single curve span that is evenly distributed over the parameter range [0, 1]. Therefore, each Bézier curve can be subdivided independently to ensure that the lengths of the linear segments approximating the original NUBS curve are the same (within a tolerance). This method has been adopted in some existing triangulation algorithms (Rockwood et al., 1988; Kumar and Manocha, 1995, 1996).

1.3.3 Geometric Subdivision

Geometric (also known as adaptive) subdivision is a method which is suitable for approximating both Bézier and NUBS since it is performed with respect to the geometric shape of the actual curve. The main aim of geometric subdivision is to only have as few linear segments as is absolutely necessary (Piegl and Tiller, 1997). Therefore, the methods generally subdivide a curve at unevenly spaced parameter intervals, which are calculated based on different geometric characteristics such as arc length, tangent angle or normal angle. Geometric subdivision is not directly used in this thesis, therefore the details have been omitted. Full details can be found in Czerkawski (1996).

1.3.4 Subdivision Overview

Uniform subdivision is simple, fast and efficient computationally, making it ideal for applications which require the subdivision of multiple curves or surfaces in real-time. A complication can occur if the NUBS curves or surfaces have to be converted into Bézier;

however this is a small price to pay for the simplicity and speed. Uniform subdivision tends to over sample a curve or surface and subdivide it into more elements than is absolutely necessary (Piegl and Tiller, 1998).

Geometric subdivision varies the approximation of the curves and surfaces depending on the local geometry reducing the number of elements required. However, it can be complicated to implement, particularly for surfaces, and is not very fast which makes it undesirable for applications which require real-time performance.

1.4 Surface Differential Geometry

Surface differential geometry can be used to gather a greater understanding of a surface's shape characteristics. This subsection will present some elements which will be used throughout this thesis.

1.4.1 Unit Normal Vector

Given a NUBS surface definition, it is often useful to know the normal direction to the surface at a given point **S**(u, v) (Section 1.4.2). The tangent vectors for the parameter curves in the u and v directions are defined as $\mathbf{S}_u(u, v)$ and $\mathbf{S}_v(u, v)$ respectively (Figure 1.11). The unit normal vector is perpendicular to both tangents vectors and is given by

$$\mathbf{n}(u,v) = \frac{\mathbf{S}_u(u,v) \times \mathbf{S}_v(u,v)}{\left\| \mathbf{S}_u(u,v) \times \mathbf{S}_v(u,v) \right\|} \tag{1.21}$$

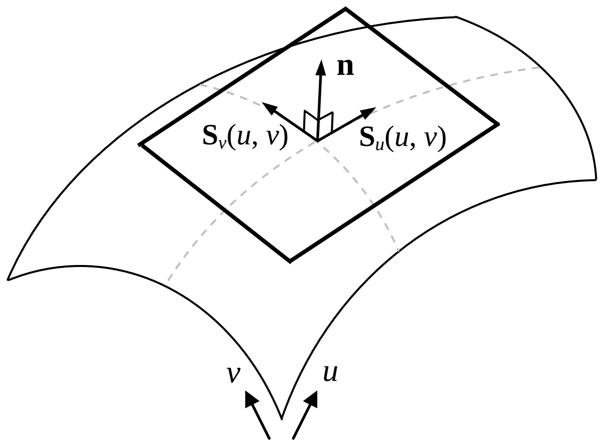


Figure 1.11 Surface normal vector.

1.4.2 Principal Curvatures and Directions

Principal curvatures can be used to provide a better interpretation of a surface's shape characteristics. At a single point on a surface, the curvature values vary depending in which direction it is being evaluated. For example, Figure 1.12 shows a cylinder surface which has unit radius. In the u direction the curvature is zero, however in the v direction the curvature is one.

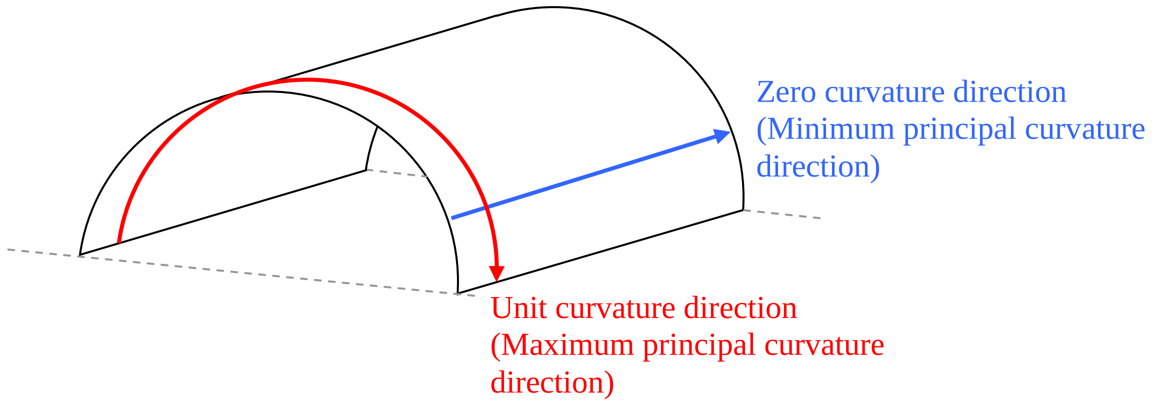


Figure 1.12 Principal curvature directions on a unit radius cylinder.

In general there is a minimum and maximum curvature value for every point on a surface, called principal curvatures, however, in general, these will not necessarily follow the u and v

directions. Therefore, each principal curvature value also has an associated direction, which is specified within the parameter domain of the surface. When mapped to 3D, the principal curvature directions will always be perpendicular to each other (Nutbourne and Martin, 1988).

The derivation of the principal curvatures and directions uses the first fundamental matrix, **G**(*u*, *v*), and second fundamental matrix, **D**(*u*, *v*) (Faux and Pratt, 1985), of the surface define as

$$\mathbf{G}(u,v) = \begin{bmatrix} \mathbf{S}_u(u,v)\cdot\mathbf{S}_u(u,v) & \mathbf{S}_u(u,v)\cdot\mathbf{S}_v(u,v) \\ \mathbf{S}_u(u,v)\cdot\mathbf{S}_v(u,v) & \mathbf{S}_v(u,v)\cdot\mathbf{S}_v(u,v) \end{bmatrix} \tag{1.22}$$

and

$$\mathbf{D}(u,v) = \begin{bmatrix} \mathbf{n}(u,v)\cdot\mathbf{S}_{uu}(u,v) & \mathbf{n}(u,v)\cdot\mathbf{S}_{uv}(u,v) \\ \mathbf{n}(u,v)\cdot\mathbf{S}_{uv}(u,v) & \mathbf{n}(u,v)\cdot\mathbf{S}_{vv}(u,v) \end{bmatrix}. \tag{1.23}$$

Let $\dot{\mathbf{u}}$ be a 2D vector with the values *du* and *dv* which are small changes in the surface *u* and *v* parameter directions respectively. Then the equation for the principal curvatures, κ_p, and directions, $\dot{\mathbf{u}}$, is defined as

$$(\mathbf{D}(u,v) - \kappa_p \mathbf{G}(u,v))\dot{\mathbf{u}} = 0 \tag{1.24}$$

or

$$(d_{00} - \kappa_p g_{00})du + (d_{01} - \kappa_p g_{01})dv = 0 \tag{1.25}$$

$$(d_{10} - \kappa_p g_{10})du + (d_{11} - \kappa_p g_{11})dv = 0 \tag{1.26}$$

where the d_{ij} and g_{ij} are the *i*th and *j*th components of the first and second fundamental form matrices respectively.

Eliminating *du* and *dv* gives

$$|\mathbf{G}(u,v)|\kappa_p^2 - (g_{00}d_{11} + d_{00}g_{11} - 2g_{01}d_{01})\kappa_p + |\mathbf{D}(u,v)| = 0 \tag{1.27}$$

From here the minimum ($\kappa_{\min}$) and maximum ($\kappa_{\max}$) curvatures can be obtained and substituted into Equations (1.25) and (1.26) to extract the ratio of du:dv for the principal directions (Faux and Pratt, 1985). For example, substituting $\kappa_{\min}$ into (1.25) and (1.26) will give

$$a_0du + a_1dv = 0 \qquad \text{and} \qquad b_0du + b_1dv = 0$$

where a_0, a_1, b_0 and b_1 are constants. Re-arranging the above will give

$$(a_0 - b_0)du = (b_1 - a_1)dv$$

which can then be used to calculate the ratio of *du*:*dv*

$$\frac{du}{dv} = \frac{(b_1 - a_1)}{(a_0 - b_0)}$$

Therefore, the principal direction for $\kappa_{\min}$ is $du = (b_1 - a_1)$ and $dv = (b_0 - a_0)$ (Figure 1.13). The principal direction for $\kappa_{\max}$ can be calculated analogously.

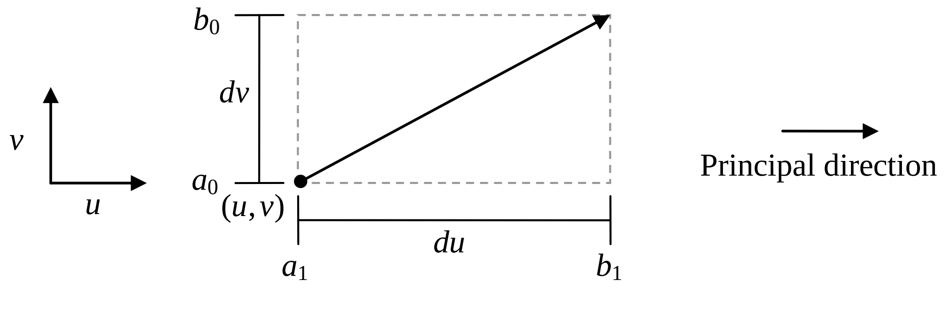


Figure 1.13 Principal direction.

1.4.3 Gaussian Curvature

The product of the two principal curvatures is known as the Gaussian curvature (Nutbourne and Martin, 1988) $K_G = \kappa_{\min}\kappa_{\max}$. It can also be expressed as a ratio between the determinants of the first and second fundamental form matrices,

$$K_G = \frac{|\mathbf{D}(u,v)|}{|\mathbf{G}(u,v)|} \tag{1.28}$$

1.4.4 Surface Curvature Classifications

Consider the local area to a point **p** on any 3D surface. The curvature on the local area can be of various types depending upon the nature of the principal curvatures $\kappa_{\min}$ and $\kappa_{\max}$ (Lipschutz, 1969). These principal curvatures will lie on mutually perpendicular planes that contain the point **p** (Figure 1.14).

A surface that has zero for one or both of the principal curvatures is called a developable. Surfaces with non-zero curvatures with different signs are called hyperbolic. Surfaces with non-zero curvatures of the same sign are called elliptical.

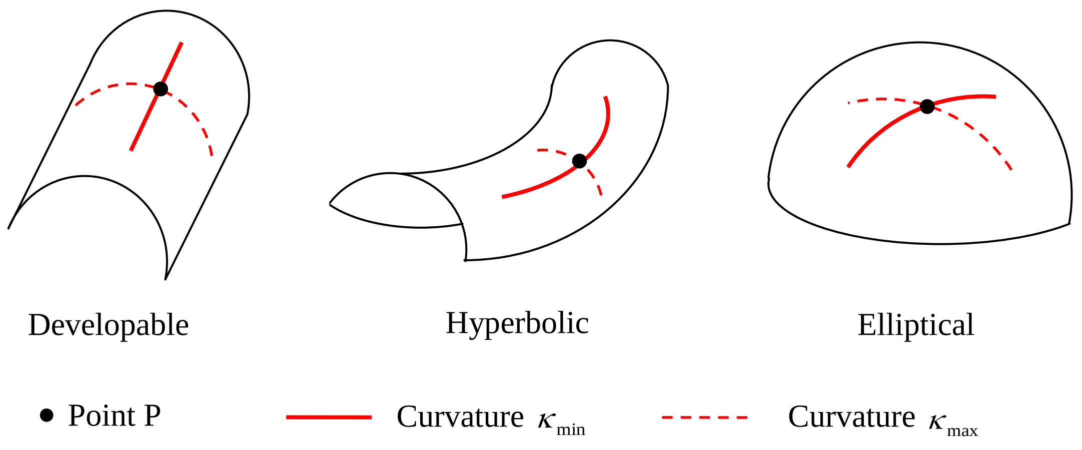


Figure 1.14 Three different types of principal curvature surface definitions.

3D surfaces that are defined as developable have the property that they can be flattened into the 2D without causing any distortion of tearing to the original shape (Rogers and Adams, 1990; McCartney *et al*., 1999). This topic will be covered in more detail in Chapter 4.

1.5 Trimmed NUBS Surfaces

A trimmed surface enables complicated 3D geometric shapes to be described using quadrilateral based NUBS surfaces. Given an underlying 3D NUBS *base surface*, $\mathbf{S}(u,v)$, consider a number of orientated trimming loops which are defined in terms of $\mathbf{S}(u,v)$. Each trimming loop consists of 2D NUBS curves, $\mathbf{C}(t)=(u(t),v(t))$, where $u(t)$ and $v(t)$ are the 2D coordinates and can be substituted into $\mathbf{S}(u,v)$ to get the 3D coordinates of the curve. The 2D curves must be fully contained within the parameter domain of the surface (Figure 1.15).

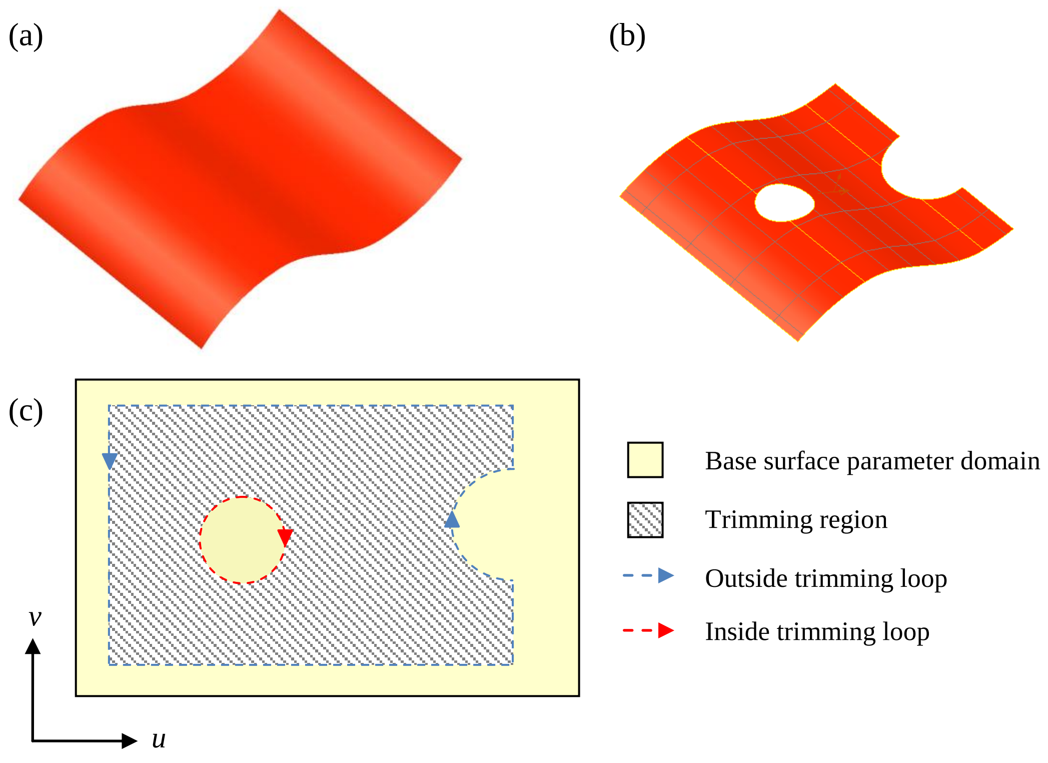


Figure 1.15 (a) Original 3D base surface. (b) Trimmed 3D surface. (c) Base surface parameter domain with the 2D trimming loops.

The purpose of the orientated loops is to define the boundaries of the surface, where the part of the surface located to the left of the loop is kept. By convention, an anti-clockwise loop (*outside loop*) defines the outside boundary of the surface, and a clockwise loop (*inside loop*) defines the inside boundaries (Kumar and Manocha, 1995, 1996). This concept of orientated loops is known as the curve handedness rule (Rockwood *et al.*,1988).

1.6 Triangle Mathematics

This subsection will introduce some triangle mathematics which will be used throughout this thesis to evaluate and operate on triangulations. In each subsection, the triangle, **t**, will be defined by the three vertices $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$. The triangle and points can be either 2D or 3D.

1.6.1 Orientation of Triangle Vertices (2D only)

The orientation of 2D triangle points is important for many applications (Schneider and Eberly, 2003). Given a 2D triangle **t**, if $\mathbf{p}_2$ is located to the left of the line with direction $\mathbf{p}_1 - \mathbf{p}_0$, then the points are in anti-clockwise. If $\mathbf{p}_2$ is located to the right of the line, then they are in clockwise order. If $\mathbf{p}_i = (x_i, y_i)$ then

$$\delta = \det \begin{bmatrix} 1 & 1 & 1 \\ x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \end{bmatrix} \tag{1.29}$$

Let *tol* be a zero tolerance value, then $\delta > tol$ means the triangle vertices are in anti-clockwise order and $\delta < -tol$ indicates a clockwise ordering. If

$$|\delta| < tol,$$

then the triangle vertices are all collinear and the triangle is degenerate (Figure 1.16). For non-degenerate triangles, the orientation of a triangle (i.e. clockwise or anti-clockwise) can be changed be reversing the ordering of the vertices. Unless otherwise stated, it will be assumed that all triangles used throughout the remainder of this thesis will have vertices in anti-clockwise order.

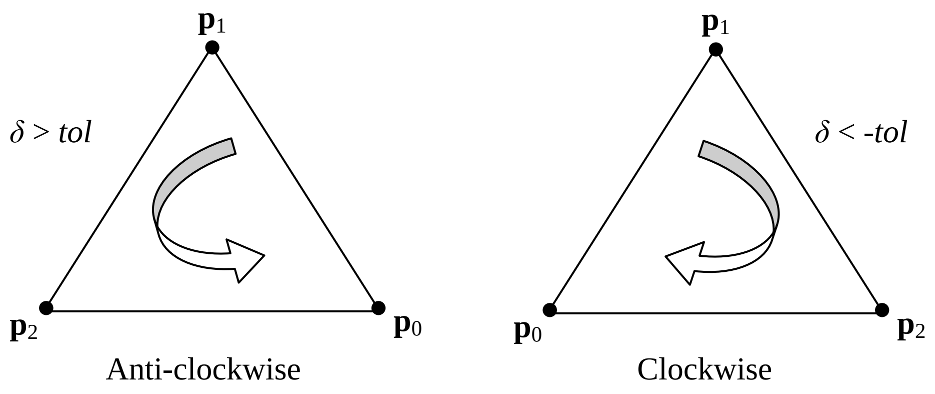


Figure 1.16 Triangle vertex orientations.

1.6.2 Point within a Triangle (2D only)

Given a 2D triangle **t** and a 2D point, $\mathbf{p}_n$, a common problem is to establish whether $\mathbf{p}_n$ is contained within **t**. Assuming that the points of **t** are in anti-clockwise order, the point $\mathbf{p}_n$ can be checked against each consecutive edge using (1.29). If $\mathbf{p}_n$ is located to the left of every edge (i.e. $\delta > tol$), then it is within the triangle (Figure 1.17).

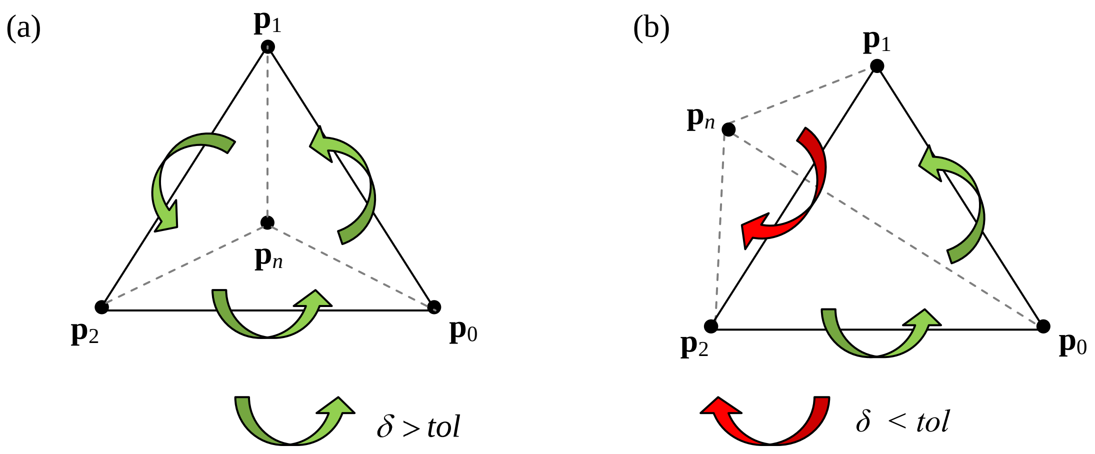


Figure 1.17 (a) $\mathbf{p}_n$ is inside the triangle as it is located to the left of every edge. (b) $\mathbf{p}_n$ is outside the triangle as it is located to the right of edge ($\mathbf{p}_1$, $\mathbf{p}_2$).

1.6.3 Triangle Centre of Gravity

The centre of gravity, $\mathbf{p}_{cog}$, sometimes referred to as the centroid, of triangle **t** is given by (O'Rourke, 2001)

$$\mathbf{p}_{cog} = \frac{\mathbf{p}_0 + \mathbf{p}_1 + \mathbf{p}_2}{3} \tag{1.30}$$

1.6.4 Triangle Perimeter

The perimeter, $\mathbf{p}_t$, of triangle **t** is given by (Schneider and Eberly, 2003)

$$\mathbf{p}_t = \left\|\mathbf{p}_0 - \mathbf{p}_1\right\| + \left\|\mathbf{p}_1 - \mathbf{p}_2\right\| + \left\|\mathbf{p}_2 - \mathbf{p}_0\right\| \tag{1.31}$$

1.6.5 Triangle Area

The area, A_t, of triangle **t** is given by (Schneider and Eberly, 2003)

$$A_t = \frac{\left\|\mathbf{p}_0 \times \mathbf{p}_1 + \mathbf{p}_1 \times \mathbf{p}_2 + \mathbf{p}_2 \times \mathbf{p}_0\right\|}{2} \tag{1.32}$$

1.6.6 Barycentic Area of a Triangle

The barycentric area, A_{bc}, of a triangle, **t**, is given by the area of the quadrilateral defined by a triangle vertex $\mathbf{p}_n$, $\mathbf{p}_{cog}$, and the mid-points of the triangle edges incident to $\mathbf{p}_n$ (Figure 1.18), which is equivalent to:

$$A_{bc} = \frac{A_t}{3} \tag{1.33}$$

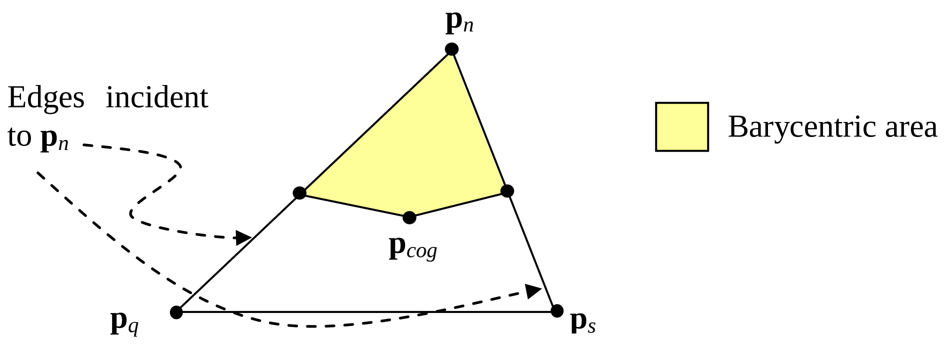


Figure 1.18 Barycentric area of a triangle.

1.6.7 Triangle Normal

Let $\mathbf{p}_a$ be the vector $\mathbf{p}_1$ - $\mathbf{p}_0$ and let $\mathbf{p}_b$ be $\mathbf{p}_2$ - $\mathbf{p}_0$ for a triangle where the vertices are in anti-clockwise order, then the normal vector, $\mathbf{n}_t$, to triangle **t** (Figure 1.19) is given by (O'Rourke, 2001)

$$\mathbf{n}_t = \mathbf{p}_a \times \mathbf{p}_b \tag{1.34}$$

where the unit normal vector, $\mathbf{n}_{ut}$, is given by

$$\mathbf{n}_{ut} = \frac{\mathbf{n}_t}{\|\mathbf{n}_t\|} \tag{1.35}$$

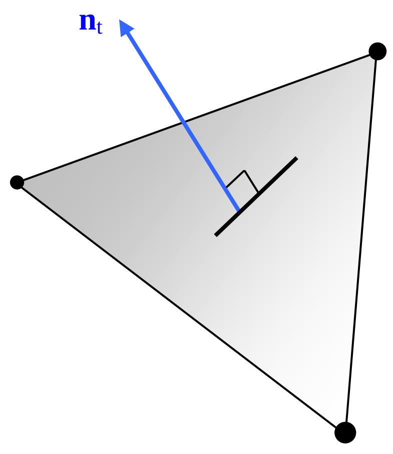


Figure 1.19 Triangle normal vector.

1.6.8 Triangle Circumcircle (2D only)s

The circle that passes through all the vertices of triangle **t** is called the circumcircle; its centre (point $\mathbf{p}_c$) is called the circumcentre and its radius, r, is known as the circumradius (Figure 1.20) (Schneider and Eberly, 2003). Let

$$a_0 = (\mathbf{p}_2 - \mathbf{p}_0) \cdot (\mathbf{p}_1 - \mathbf{p}_0), \quad a_1 = (\mathbf{p}_2 - \mathbf{p}_1) \cdot (\mathbf{p}_0 - \mathbf{p}_1), \quad a_2 = (\mathbf{p}_0 - \mathbf{p}_2) \cdot (\mathbf{p}_1 - \mathbf{p}_2)$$

$$b_0 = a_1 a_2, \quad b_1 = a_2 a_0 \quad \text{and} \quad b_2 = a_0 a_1 .$$

Then

$$r = \sqrt{\frac{(a_0 + a_1)(a_1 + a_2)(a_2 + a_0)}{2(b_0 + b_1 + b_2)}} \tag{1.36}$$

$$\mathbf{p}_c = \frac{(b_1 + b_2)\mathbf{p}_0 + (b_2 + b_0)\mathbf{p}_1 + (b_0 + b_1)\mathbf{p}_2}{2(b_0 + b_1 + b_2)} \tag{1.37}$$

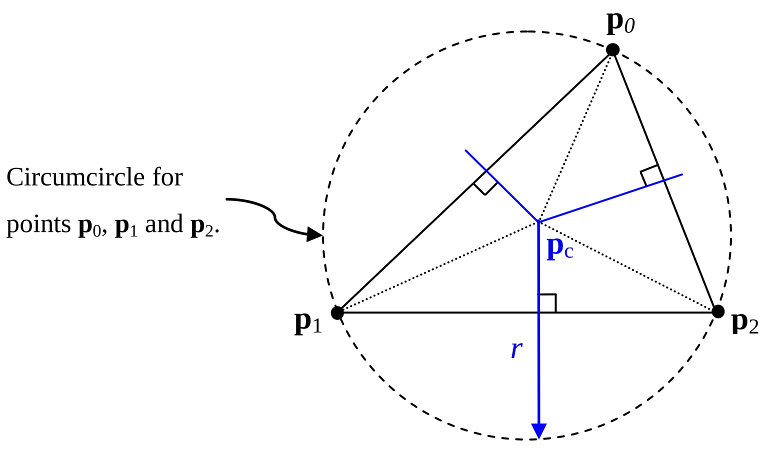


Figure 1.20 Triangle and its circumcirle.

1.6.9 Triangle Aspect Ratio

The aspect ratio of a triangle is ratio between the longest edge length and the smallest amplitude. Given a triangle, **t**, with vertices $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$, the edge lengths can be calculated using

$$l_0 = \|\mathbf{p}_1 - \mathbf{p}_0\|, \quad l_1 = \|\mathbf{p}_2 - \mathbf{p}_1\|, \quad l_2 = \|\mathbf{p}_0 - \mathbf{p}_2\| \tag{1.38}$$

The longest edge is the one with the longest length, l_{long} (l_2 in Figure 1.21).

The smallest amplitude will always be perpendicular to the longest edge. This amplitude, a_{small}, is calculated by taking an edge length other than l_{long}, say l_1 in Figure 1.21, and then $a_{small} = l_1\cos\alpha$ where $\alpha = \theta$-90º.

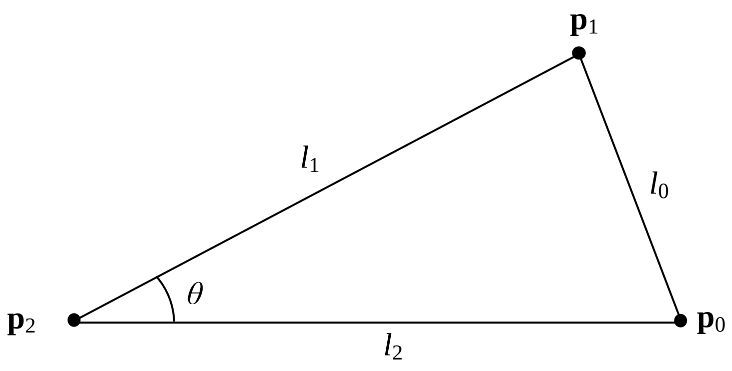


Figure 1.21 Aspect ratio of a triangle.

Therefore, the aspect ratio, a_{ratio} is given by

$$a_{ratio} = \frac{l_{long}}{a_{small}} \tag{1.39}$$

1.7 Triangulation Representation

In general, triangulations of surfaces have a large number of triangles which need to be stored in a robust manner and operated on efficiently. To make this possible, triangulations are generally stored in lists of their topological elements, namely vertices, edges and triangles. Whilst compiling these lists, an additional measure is taken to establish and store the relationships between the topological elements. For example, each edge will store pointers to the two vertices that define it and two pointers to the faces that are incident to it.

The relationships between the topological elements are listed below:

1. Each vertex has *n* pointers to incident edges and *n* pointers to incident triangles.
2. Each edge has 2 pointers to the vertices define its end points and 2 pointers the triangles incident to this edge.
3. Each triangle has 3 pointers to the vertices and 3 pointers to the edges that define it.

By storing a triangulation in this relational manner, queries such as "which triangles are incident to a desired vertex" can be executed efficiently without incurring the overhead of having to search through all the topological lists to calculate the answer, thus increasing data retrieval efficiency.

To improve the robustness and consistency of data, it is also required that every triangle has a consistent orientation of points, i.e. anti-clockwise or clockwise. Conventionally, anti-clockwise orientation of triangle points is used as this ensures that the normal to a triangle face will point outwards, which is useful for downstream task such as creating lighting effects in visualisations.

1.8 Thesis Structure and Objectives

The remaining chapters of this thesis, and their contents, will be as follows:

Chapter 2 will present a literature review on existing surface triangulation methods and approaches. It will begin fairly general and then be more focused towards trimmed surface triangulation methods using a tracing approach. The reason for this is because triangulation of a trimmed surface is more complex, which has resulted in this particular area of research being more active and producing many different algorithms.

Chapter 3 will present a detailed description of the trimmed surface triangulation scheme developed, in particular focusing attention of how trimmed surfaces are traced efficiently by only searching through cells that intersect the trimming loops (Cripps and Parwana, 2011).

Chapter 4 will give an introduction into surface flattening. It will give a review of the McCartney *et al*. (1999) and Wang *et al*. (2002) methods before presenting the Geometric Modelling Group (GMG) flattening scheme (Cader *et al*., 2005), which was derived from these original methods. The GMG method produces an approach for determining the positions of 3D points in 2D which is equivalent to the intersection of circles methods presented by both McCartney *et al*. and Wang *et al*.. This corresponding method allows for runtime modifications of the flattened triangulation making the method more efficient.

Chapter 5 will introduce the concept of right angle triangle configurations (RATCs) within a triangulation. The objectives here will be established whether different the RATCs, generated from the same point sampling, will change the final flattening results, and if so, which RATC would be the best overall choice in terms of algorithm stability and generating minimal distortion. To investigate this, it will introduce three global configurations and then demonstrate the effects of each configuration on surface flattening (Parwana and Cripps, 2009). A new method for generating a RATC, that changes locally based on the Gaussian curvature approximation, will be introduced. In addition, the method for approximating the Gaussian curvature from a triangulation will be improved to deal with varying vertex valencies within a triangulation. Furthermore, the three global and Gaussian curvature RATCs will be tested on the application of flattening using two industrial surfaces.

Chapter 6 will show that having a developable surface does not intrinsically imply that the surface triangulation of that surface will be developable. This can have interesting implications on the result of a surface flattening by inducing distortion (Parwana and Cripps, 2009). Identifying this problem, the objective of this chapter will be to establish a new method for generating a triangulation from a surface that will attempt to maximise the inheritance of developability (shape characteristic) from the initial surface.

Chapter 7 will conclude the thesis and provide possible avenues to future work.

Chapter 2

SURFACE TRIANGULATION

There are a large number of algorithms which have been developed for triangulating a surface. These can be categorised as structured and unstructured (Frey and George, 2000). A structured triangulation is an approximation that is generally uniform and has a noticeable pattern, where as an unstructured triangulation is the opposite. An example of each can be seen in Figure 2.1.

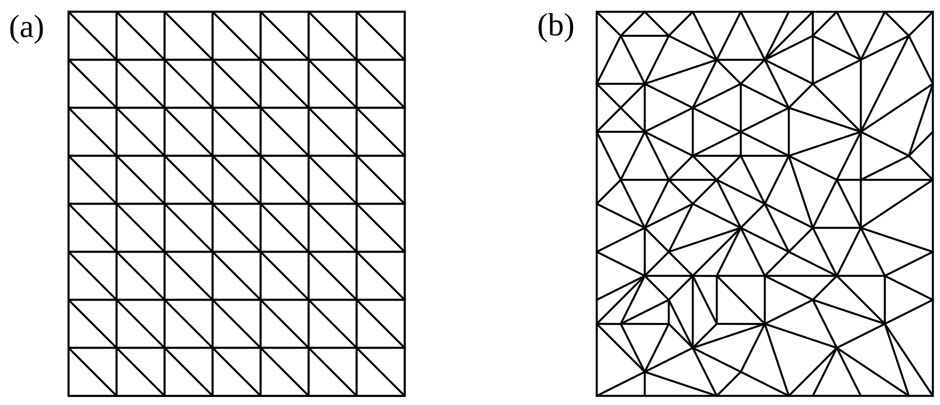


Figure 2.1 (a) Structured and (b) unstructured triangulation.

This chapter will review a variety of triangulation generation algorithms which can be used to triangulate trimmed surfaces. Trimmed surfaces are complex geometries and the triangulation of these surfaces is a non-trivial task, thus encouraging much research leading to many algorithms. The main aim of this thesis is to investigate the downstream effects imposed on flattening by triangulations that are structured. Structured triangulations are commonly used for surface flattening (McCartney *et al*., 1999; Wang *et al*., 2002; Cadar *et al*., 2005; Parwana and Cripps, 2009) and provide some interesting degrees of freedom, which have not been explored. Investigating these variables should provide valuable insight into how subtle changes in a triangulation can affect the downstream method of surface flattening. This will be discussed in more detail in Section 2.10.

For completeness, unstructured triangulation generation methods will also be introduced in this chapter. Sections 2.1 to 2.9 will present the different triangulation algorithms and Section 2.10 will discuss the differences, advantages and disadvantages of the methods.

A common requirement of many algorithms is the ability to triangulate polygons and sets of points. As these form the basis of many triangulation algorithms, these are introduced first.

2.1 Polygon Triangulation

A classic problem in computer graphics is to decompose a simple polygon into a set of triangles. Every simple polygon admits a triangulation, where a simple polygon with *n* vertices consists of exactly *n*-2 triangles (de Berg *et al*., 2000). Although there are many different algorithms available for polygon triangulation (Clarkson *et al*., 1989; Seidel, 1991; de Berg *et al*., 2000; O’Rourke, 2001), the ear clipping method is presented due to it relative simplicity.

2.1.1 Simple Polygon definition

By definition, a simple polygon is an ordered sequence of n vertices $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$. It is assumed that the vertices are orientated in anti-clockwise order. Each consecutive pair of vertices is connected by an edge $\vec{\mathbf{e}} = (\mathbf{p}_i, \mathbf{p}_{i+1})$, (where i is treated as modulo n, i.e. $x_n = x_0$). A simple polygon edge can only intersect at the vertices and each vertex is incident to exactly two edges (Figure 2.2).

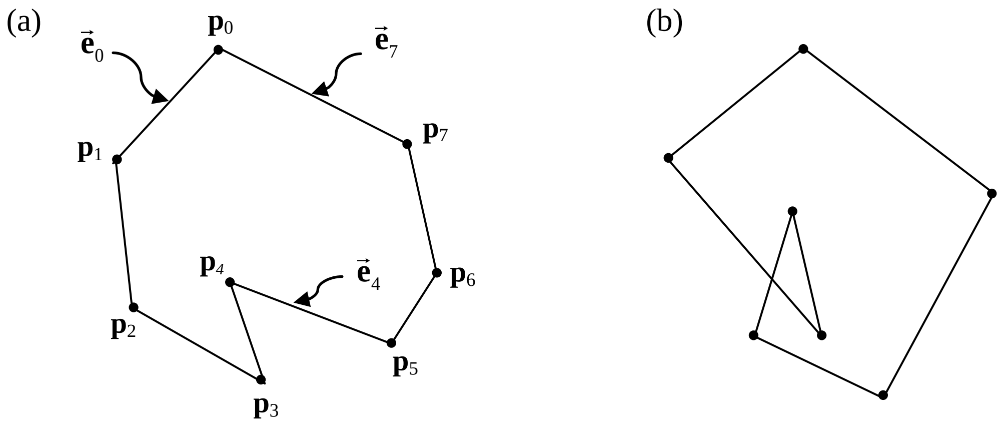
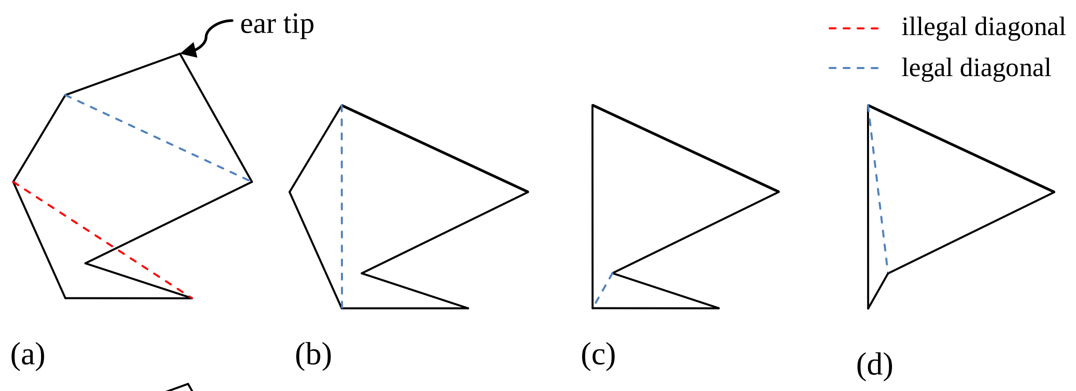


Figure 2.2 (a) Simple polygon. (b) Non-simple polygon.

2.1.2 Ear Clipping

An ear of a polygon is a triangle formed by consecutive vertices $\mathbf{p}_i$, $\mathbf{p}_{i+1}$, and $\mathbf{p}_{i+2}$ where no other polygon vertex lies within the triangle. Vertex $\mathbf{p}_{i+1}$ is the ear tip, and the line joining vertices $\mathbf{p}_i$ and $\mathbf{p}_{i+2}$ is called a diagonal (Figure 2.3(a)). Once an ear is found, the triangle is constructed by inserting a diagonal and removing the ear tip, forming a new polygon with n-1 vertices. This process is continued until only three vertices remain. Due to the nature of this method, the triangulation generated is not necessarily unique, as a different starting vertex could remove ears in a different order, thus altering the final triangulation. Figure 2.3, shows a working example of this method performed on a polygon with $n = 7$ vertices.



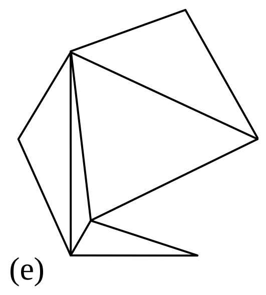


Figure 2.3 (a) A polygon with a legal and illegal diagonal. (a) - (d) Ear clipping process and (e) final triangulation.

2.2 2D Delaunay triangulation

Another recognised problem in computer graphics is to generate a non-intersecting set of triangles from a set of n 2D points $\mathbf{P} = \mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$, where the points of **P** form the vertices of the triangles. This can be achieved by using the Delaunay triangulation technique, which was first proposed by Boris Delaunay in 1934 (de Berg *et al*., 2000).

2.2.1 Circumcircle Property

The fundamental property of a planar Delaunay triangulation is that the circumcircle of any triangle, within the triangulation, may have no other points of the set **P** within it, apart from the points scribing the circumcircle (Watson, 1981) as shown in Figure 2.4. Due to this property, the Delaunay triangulation maximises the minimum angle of any triangle. In addition, the boundary edges of a Delaunay triangulation form the convex hull of the original points set **P** (O'Rourke, 2000). Furthermore, the Delaunay triangulation of any set of points **P** will produce a unique solution (de Berg *et al.*, 2000; O'Rourke, 2001; Schneider and Eberly, 2003). This statement will be explored in Section 2.10.

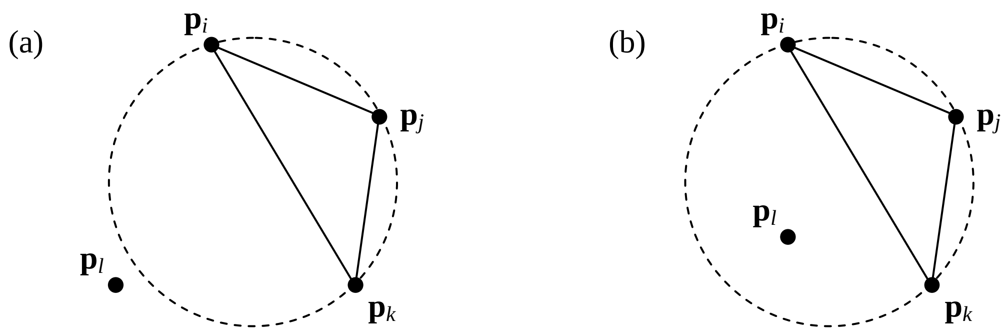


Figure 2.4 (a) Legal and (b) illegal triangle within a Delaunay triangulation.

2.2.2 Algorithms

Many algorithms exist for the construction of a Delaunay triangulation from a set of points. A direct implementation uses a divide and conquer or randomised incremental approach (de Berg *et al.*, 2000; O'Rourke, 2001; Schneider and Eberly, 2003). However, as unstructured triangulation methods are quite involved and not the direct subject of this thesis the details of their implementations have been omitted. Full details can be found in Schneider and Eberly (2003).

2.2.3 Delaunay Triangulation and Trimmed Surfaces

Although the Delaunay triangulation method was originally created to triangulate an unstructured set of points, the method has been adopted into some existing algorithms for generating a triangulation from a trimmed surface. Sheng and Hirsch (1992) present a method for triangulating a trimmed parametric surface, based on partitioning the parameter space and a Delaunay triangulation. First, a maximum 2D triangle edge size, Ω, is determined by using a 'flatness criteria' (Peterson, 1994). This ensures that when the triangle is mapped into 3D it does not deviate from the surface by more than some user specified tolerance. Once the surface and trimming curves have been subdivided, the points located within the trimming region are identified and triangulated using a Delaunay triangulation.

Piegl and Richards (1995) generalised this method for trimmed NURBS surfaces. In addition, the points located within the trimming region are established by using a simple scan-line-type algorithm, as used in raster graphics to fill polygons (Foley *et al*., 1995).

As the Delaunay triangulation of surfaces is complex and not the main focus of this thesis, full details of existing algorithms have been omitted. However, a comprehensive survey of existing methods for triangulating a trimmed surface using a Delaunay triangulation can be found in Frey and George (2000).

2.3 Advancing Front Method

The advancing front method (AFM) is used to generate an unstructured triangulation from surfaces which may or may not be trimmed.

2.3.1 Boundary Edge Subdivision

The method begins by defining the 2D exterior boundary edges of the trimming region, within the parameter domain of the surface, and subdividing them geometrically into piecewise linear segments. This ensures that the linear segments will have the same length, within a user defined tolerance, when mapped into 3D. These 2D piecewise linear boundary edges are called the front (Figure 2.5).

2.3.2 Point Generation from the Current Front

The next stage is to generate new points, within the parameter domain of the surface, towards the interior of the trimming region from the current front (Figure 2.5). Once generated, these new interior points will be connected with the points of the front to form a list of triangles (Figure 2.5(b)). In addition, the piece-wise linear curve produced by inscribing the interior points generated will form the new current front.

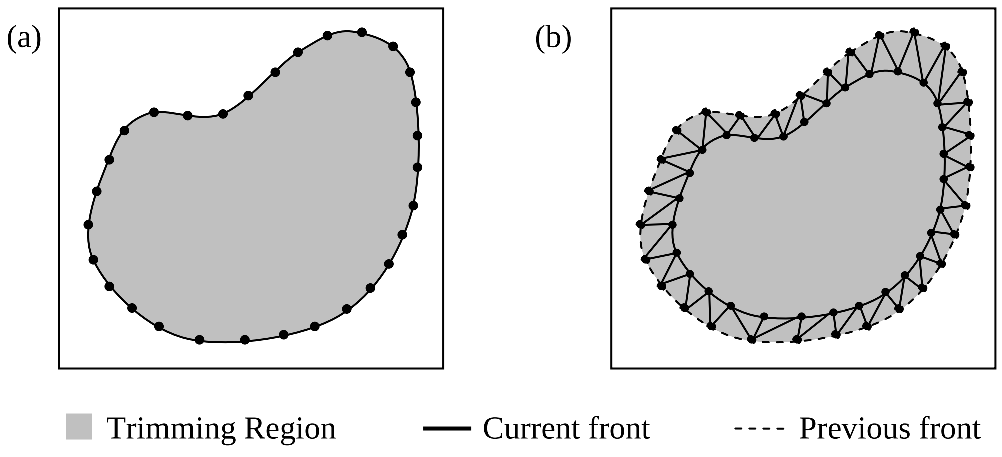


Figure 2.5 Trimming region of a surface parameter domain. (a) Initial front. (b) New front and triangles generated after one iteration.

The point generation process is accomplished by taking a linear segment from the front, with end points $\mathbf{p}_0$ and $\mathbf{p}_1$, and then determining an optimum position for $\mathbf{p}_2$ such that the edge lengths and interior angles of the triangle ($\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$) are the same, within a user specified tolerance, when mapped back into 3D (Frey and George, 2000). On determining the position of $\mathbf{p}_2$, edge ($\mathbf{p}_0$, $\mathbf{p}_1$) is removed from the current front and edges ($\mathbf{p}_0$, $\mathbf{p}_1$) and ($\mathbf{p}_1$, $\mathbf{p}_2$) are added

(Figure 2.6), thus updating the current front. This process of generating triangles and updating the front is repeated until the interior region has been completely covered by triangles.

2.3.3 Algorithms

As with Delaunay triangulation, many algorithms exist for the construction of an AFM triangulation from a surface. The main difference between the algorithms is the point generation process, which is largely dependant on the geometric criteria provided for the triangle shapes. A comprehensive account of many AFM methods can be found in Frey and George (2000) for trimmed surfaces and Wu and Wang (2005) for triangulating a regular parametric surface.

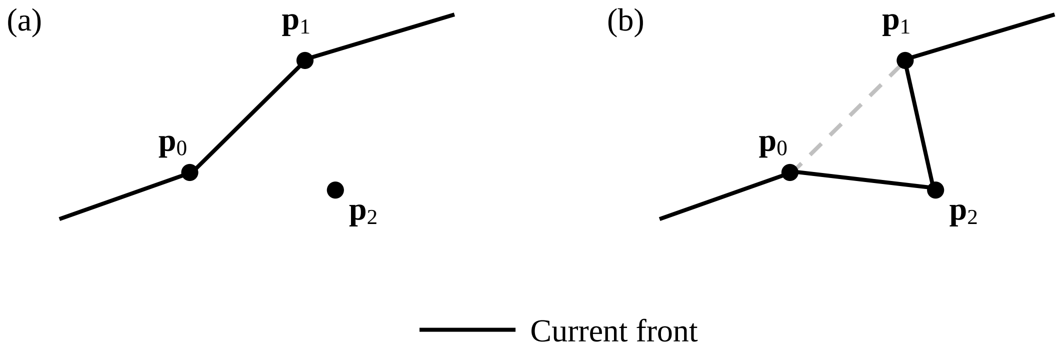


Figure 2.6 (a) New point generated from current front. (b) Updated front.

2.4 Trimmed Surface Structured Triangulation Methodology

The general methodology adopted by many existing algorithms for generating a structured triangulation from a trimmed surface is very similar. Therefore, to aid in understanding the triangulation algorithms presented in this chapter, a generic process is presented.

Parameter domain triangulation

Trimmed surface triangulation algorithms generally operate within the 2D parameter domain of the NURBS surface. This considerably reduces the complexity of the problem, and is convenient as the trimming loops are defined in the 2D parameter plane of the surface.

Subdivision of the parameter domain

The surface parameter domain is subdivided, either uniformly or geometrically, into a grid of points. In some instances, the points are strategically generated to construct a grid of rectangular cells. The NURBS surface definition is used to determine the number of intervals that the parameter domain should be subdivided into and is governed by its characteristics such as tangent, curvature or flatness.

Approximating the curves

The trimming curves are subdivided into linear segments either uniformly or geometrically. The number of subdivisions for the NURBS curve is governed by its characteristics such as tangent and curvature.

Identifying points/cells with trimming region

The linear approximations of the trimmed curves are superimposed onto the points/cells to establish whether they are inside or outside of the trimming region, within a user specified tolerance. Essentially, if a point (cell) is fully located within an outside loop and is not contained within any inside loop, then that point (cell) is within the trimming region. Points (cells) within the trimming region are stored for triangulation, while the others are discarded.

Cells partially within trimming region

When using cells, many will be located partially within the trimming region. This is because a trimming curve will intersect and pass through it (Figure 2.7). Therefore, the cells located along the boundary edges of the trimming regions require a little more attention before they can be triangulated. Cell based algorithms record the intersections between cells/trimming loops and then convert these boundary edge cells into polygons (Figure 2.7). These polygons are then stored for triangulation, along with the cells completely within the trimming region.

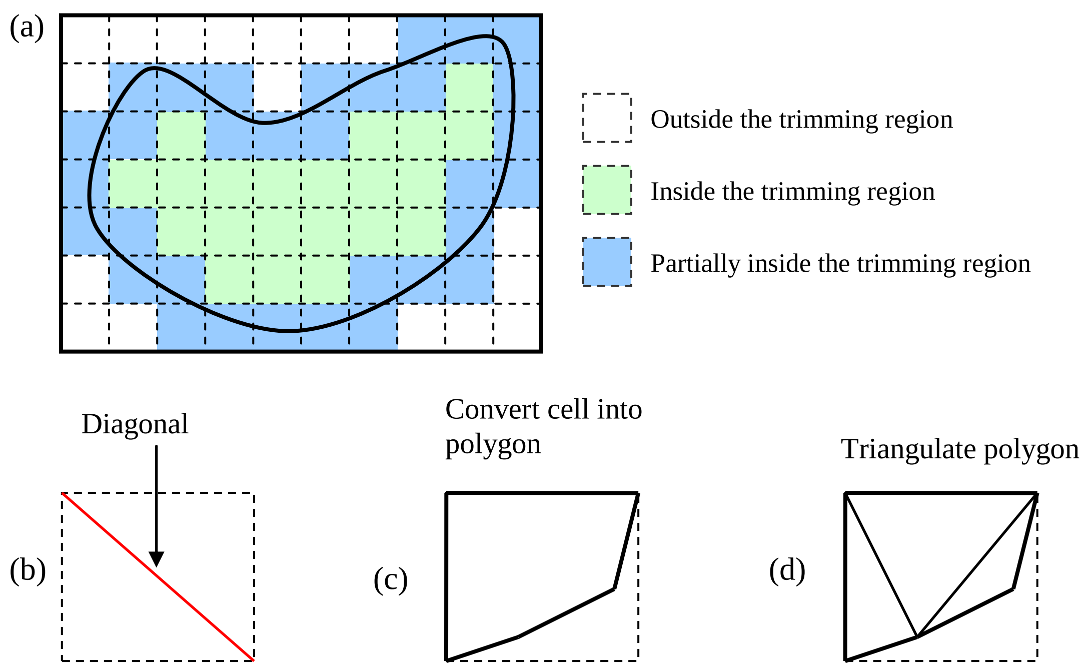

(a)
Outside the trimming region
Inside the trimming region
Partially inside the trimming region
Diagonal
Convert cell into polygon
Triangulate polygon
(b)
(c)
(d)

Figure 2.7 (a) Cell classification within the parameter domain of a surface. Methods used for triangulation (b) inside and (c)-(d) partially inside cells.

Triangulation

If working with points, the 2D parameter domain points are triangulated using the Delaunay triangulation technique.

When working with cells, the stored polygons are triangulated to form an overall 2D triangular representation of the initial model. The triangulation of a polygon with four points (e.g. all cells completely contained within the trimming region) is a special case that is simplified by adding a diagonal (Figure 2.7(b)). It is worth noting that it is around the boundary edges of a trimmed surface that the triangle shapes can become non-uniform (unstuructured) (Figure 2.7(d)).

Mapping back to 3D

The final stage of a triangulation algorithm is to map the 2D triangulation into 3D. This is done by taking the 2D coordinates of triangle vertices, in terms of *u* and *v*, and substituting them into the original 3D surface definition to get the corresponding 3D coordinates.

Sections 2.5 to 2.9 will each present a different structured triangulation algorithm that utilises this general methodology to triangulate a trimmed NURBS surface. The triangulation methods will be presented in chronological order.

2.5 Rockwood *et al*. triangulation

Rockwood *et al*. triangulation (Rockwood *et al*., 1989) was developed to render trimmed surfaces in real-time, making it ideal for graphical visualisation. The method produces a view-driven triangulation of a trimmed surface, altering the density of the triangulation depending on its distance from the current view point.

Converting to NURBS to Bézier

The first stage of the algorithm is to convert the initial NURBS surface and curves into rational Bézier form using knot insertion (Section 1.2.3). The trimming loops must also be subdivided along the new patch boundaries to maintain a valid trimmed surface representation

(Section 1.5). Therefore, additional curves are added along the patch boundaries, to restrict the trimming region to individual patches (Figure 2.8).

Once the trimmed surface is fully converted, each patch is triangulated independently. The lists of triangles are then merged into a single list to produce an overall triangulation of the initial surface.

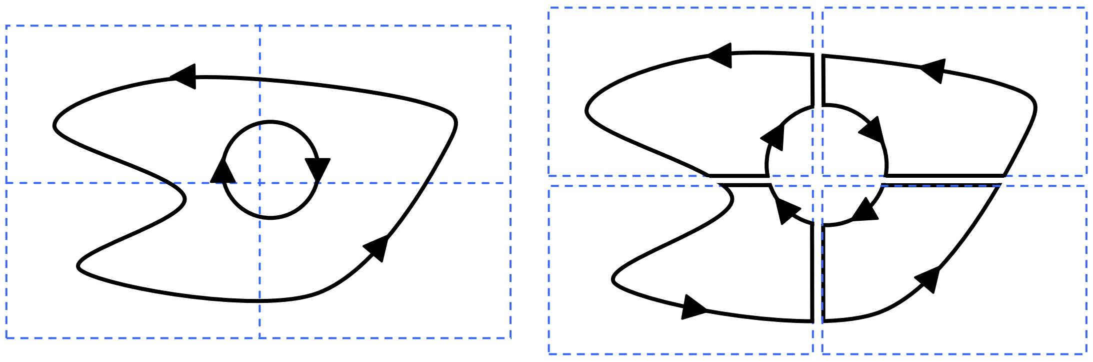


Figure 2.8 (a) Original parameter domain and trimming cuvres. (b) New parameter domains post Bézier subdivision.

Subdividing Surface and Trimming Loops

Each patch is subdivided into a rectangular grid, and each trimming curve is decomposed into piecewise linear segments so that the edge lengths of the triangles do not exceed a user specified tolerance in 3D space.

Let π_3 be the projection from image space to screen space for a $(n, m)^{th}$ degree rational Bézier patch with 3D control points $\mathbf{P}_{i,j}$ and weights $w_{i,j}$, then the number of steps in the u and v directions (between 0 and 1), n_u and n_v, which will guarantee triangle edge lengths in screen space are less than a specified tolerance, *tol*, (Rockwood *et al*. 1989) are given by

$$n_u = \frac{n\sqrt{2}\max\ \left\| w_{i,j}\pi_3\mathbf{P}_{i,j} - w_{i+1,j}\pi_3\mathbf{P}_{i+1,j} \right\|}{tol \times \min(w_{i,j})} \qquad \begin{matrix} 0 \le i \le n-1 \\ 0 \le j \le m \end{matrix} \tag{2.1}$$

$$n_v = \frac{m\sqrt{2}\max\ \left\| w_{i,j}\pi_3\mathbf{P}_{i,j} - w_{i,j+1}\pi_3\mathbf{P}_{i,j+1} \right\|}{tol \times \min(w_{i,j})} \qquad \begin{matrix} 0 \le i \le n \\ 0 \le j \le m-1 \end{matrix} \tag{2.2}$$

The parameter increment, which will guarantee a screen space steps less than *tol*, is given by

$$step_u = \frac{1}{n_u} \quad \text{and} \quad step_v = \frac{1}{n_v} \tag{2.3}$$

The number of steps n_t that a p^{th} degree trimming curve, with control points $\mathbf{P}_i$ and weights w_i, has to be partitioned into to ensure that each linear segment in screen space is less than $step_u$ or $step_v$, respectively, is given by

$$n_t = \frac{p\max\ \left\| w_i\mathbf{P}_i - w_{i+1}\mathbf{P}_{i+1} \right\|}{step_u \times \min(w_i)} \qquad 0 \le i \le p-1 \tag{2.4}$$

$$n_t = \frac{p\max\ \left\| w_i\mathbf{P}_i - w_{i+1}\mathbf{P}_{i+1} \right\|}{step_v \times \min(w_i)} \qquad 0 \le i \le p-1 \tag{2.5}$$

Rockwood *et al*. (1989) state that if the curve is horizontal and only increments in *u*, then (2.4) is to be used. If the curve is vertical and only increments in *v*, then (2.5) is to be used. For all other curves, the maximum of (2.4) and (2.5) is to be used. Therefore, the parameter step size for the curve, $step_t$, is given by

$$step_t = \frac{1}{n_t} \tag{2.6}$$

UV Monotone trimming regions

The next stage is to calculate the maxima and minima (extrema points) of the trimming curves, which are then used to split the piecewise curves into *uv* monotone segments (Rockwood *et al*., 1988). Vertical and horizontal lines can then be extended and intersected from these points to form *uv* monotone regions within the surface (Figure 2.9). This pre-processing step helps to speed the actual triangulation process.

Tiling and Coving

The final stage of the algorithm is to superimpose a lattice of points a distance of $step_u$ and $step_v$ apart over the *uv* monotone region, and segmenting the trimming curves into points which are $step_t$ apart. A fast and simple triangulation method called tiling and coving is used (Rockwood *et al.*, 1988) and is made possible as the trimming regions are *uv* monotonic. The triangulation is performed in strips of cells, where a *v*-strip and a *u*-strip are represented as shown in Figure 2.10.

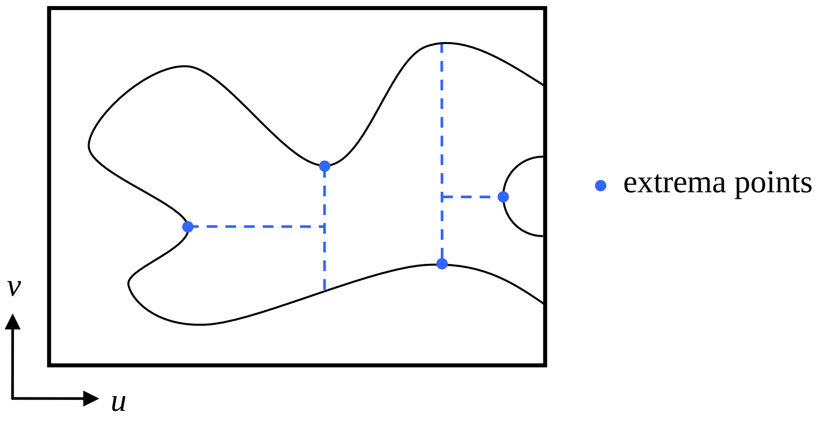


Figure 2.9 Converting the trimming region into *uv* monotone sections.

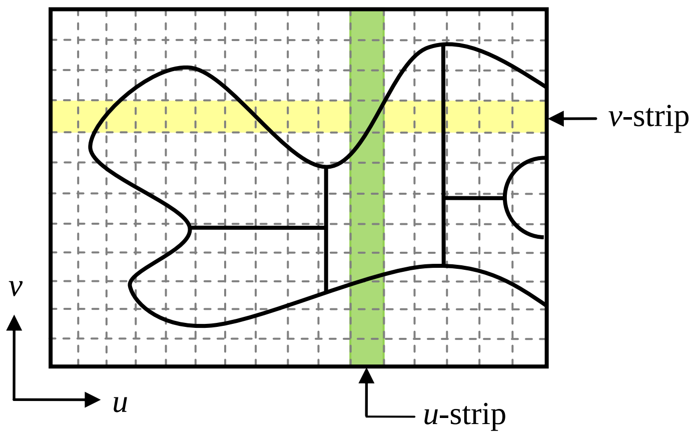


Figure 2.10 Triangulation is performed in strips.

2.6 Kumar and Manocha triangulation

Kumar and Manocha (1995) present a triangulation scheme which is optimised for the rendering of trimmed surfaces. This algorithm introduced a technique which traces the trimming loops onto the surface, thus reducing the amount of pre-processing.

Converting NURBS to Bézier

First the original NURBS surface and trimming loops are converted into rational Bézier form using knot insertion.

Subdividing Surface and Trimming Loops

Each rational Bézier patch is uniformly subdivided into a grid of rectangular cells, with the step sizes in the *u* and *v* directions, n_u and n_v, given by

$$n_u = \frac{\sqrt{2}\left\|\max(\mathbf{S}_u(u,v))\right\|}{tol}, \tag{2.7}$$

$$n_v = \frac{\sqrt{2}\left\|\max(\mathbf{S}_v(u,v))\right\|}{tol}, \tag{2.8}$$

where *tol* is a user specified tolerance for the maximum triangle size and $\max(\mathbf{S}_u(u,v))$ corresponds to the maximum partial derivative with respects to *u* in the domain [0,1] × [0,1]. n_v is calculated analogously (Kumar and Manocha, 1996).

The trimming curves are polygonised into linear segments similarly, with the parameter step size, n_t, given by

$$n_t = \frac{\sqrt{2}\left\|\max(\mathbf{C}_t(t))\right\|}{tol}, \tag{2.9}$$

where $\mathbf{C}_t(t)$ is the derivative of the curve with respects to *t*.

Tracing the Trimming Loops

Each trimming curve is then traced onto the rectangle *uv*-cells. This is achieved by systematically working through the trimming loop linear segments and recording any intersections that occur between them and the cell edges. Each cell is then labelled as being inside (active), outside (inactive) or partially inside (partially active) within the trimming region (Figure 2.11). There are two categories of partially active cells, known as prime-active and pseudo-active. A cell is prime-active when there are points of the trimming loops contained within it and pseudo-active when the cell contains no points. Each strip of cells is then evaluated to establish whether they are active or inactive, where an inactive strip is one where all the cells within it are classified as *outside*.

In addition, each end point of a curve line segment that causes an edge crossing is labelled to indicate whether it is an exit or entry point (Figure 2.11).

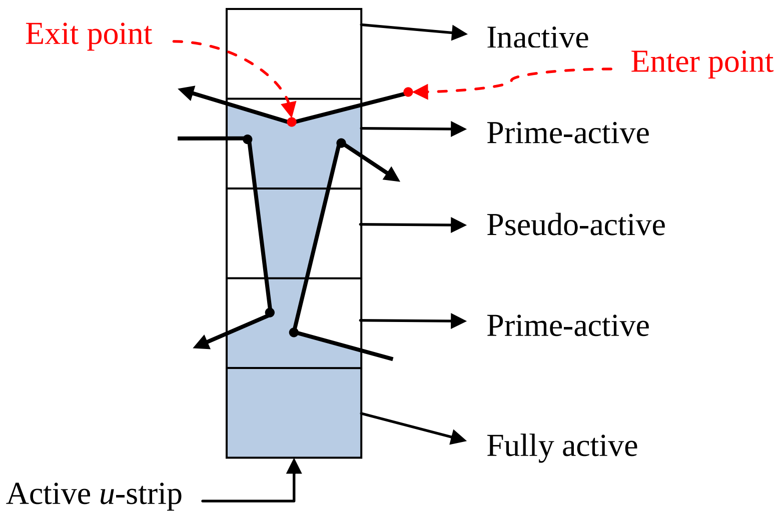


Figure 2.11 Cell labelling scheme adopted by Kumar and Manocha.

Converting Boundary Edge Cells to Polygons

Once the tracing process has been completed, the next stage is to merge any intersected cells with the trimming curves and create polygons around the boundary edges of the trimming region (Figure 2.12).

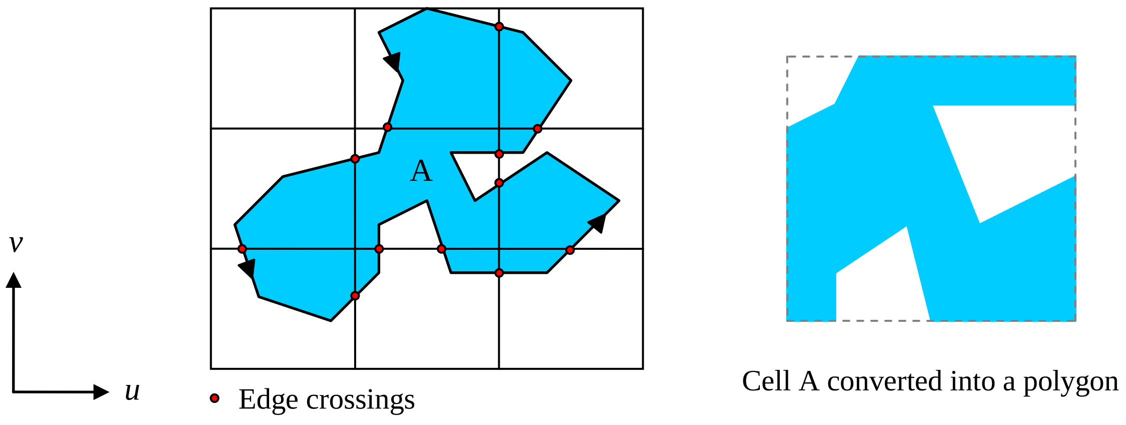


Figure 2.12 Converting cell A into a polygon.

Triangulation of cells

Triangulation of the cells is performed in strips. Each new strip is searched to check whether it is active or inactive. For active strips, any cells that have been converted into polygons are triangulated using polygon triangulation. Inactive strips are skipped as no cells within them require triangulation, thus helping to increase the efficiency of this triangulation step.

2.7 Luken Triangulation

Luken (1996) presents an algorithm which triangulates a trimmed NURBS surface directly rather than decomposing it into Bézier and triangulating each patch independently.

Working with NURBS Directly

The algorithm begins by subdividing the base NURBS surface into a global grid of cells. Although no mathematical details are provided, Lukens suggests that three separate step size calculations be performed on the surface which will generate multiple step sizes, n_u and n_v, for the u and v directions respectively. Once the step sizes have been determined, the largest n_u and n_v will be used to subdivide the entire NURBS surface. The cells generated will guarantee to produce triangle sizes that meet a user specified criteria, throughout the whole NURBS surface, when mapped to 3D space.

The suggested calculations for the step sizes are:

1. The number of steps for each patch using a calculation similar to that of Rockwood (1989) or Kumar and Manocha (1995).
2. The number of steps based on the chord length criterion (Piegl and Tiller, 1997).
3. The number of steps based on the chordal deviation criterion (Piegl and Tiller, 1997).

The subdivision of the trimming curves is performed in the same manner as for the surface.

Clipping Strips of Cells

The next stage of the algorithm is to place the linearly approximated trimming loops onto the cell grid and calculate intersections between them. This process is performed in strips of cells. A v-strip is defined as a horizontal strip of cells with the same minimum and maximum v coordinates. A u-strip is defined analogously. The method involves clipping the whole grid to one v-strip, and then clipping the v-strip to a u-strip so that essentially only one cell is being operated on (Figure 2.13).

The intersections between this cell and any trimming loops are recorded. If no intersections occur, the process is repeated on the next *u*-strip. If any intersections do occur, the cell is converted into a polygon and immediately triangulated using any polygon triangulation algorithm (e.g. ear clipping presented in Section 2.1 would be acceptable) and stored. When all the *u*-strips have been operated on, the grid is then clipped to the next *v*-strip and the whole process repeats until no *v*-strips remain.

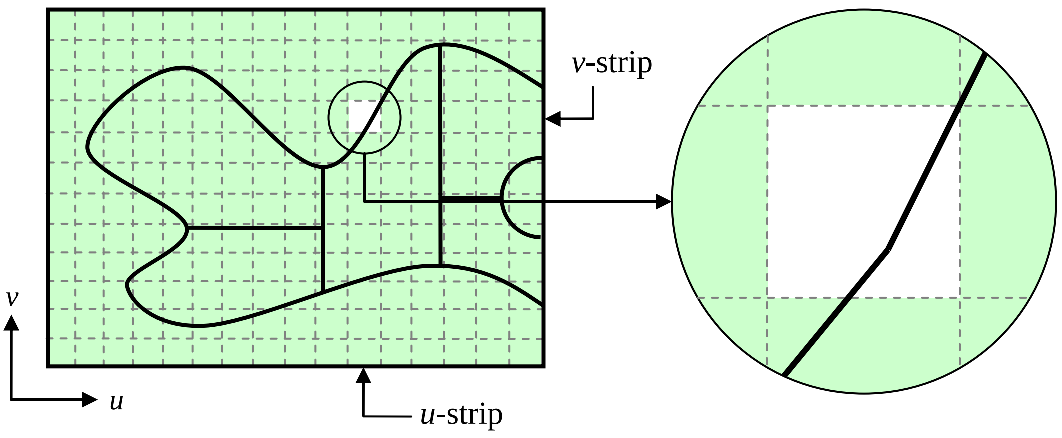


Figure 2.13 Clipping the uv-strips.

2.8 Piegl and Tiller Triangulation

The algorithm presented by Piegl and Tiller (1998) performs a trimmed surface triangulation based on the geometry of the initial shape.

Adaptive Subdivision of Curves and Surfaces

This method, like Lukens (1996), operates on the NURBS directly and uses an adaptive subdivision of the curves and surfaces based on their geometries. The trimming loops are subdivided using a 4-point cubic Bézier curve fitting technique and the surfaces are subdivided using a flatness algorithm, presented by Peterson (1994).

Modified Tracing Approach

The method uses a modified tracing scheme of Kumar and Manocha (1995, 1996). It requires that cells of the grid are labelled as one of five different categories (Figure 2.14):

1. *IN*: Cell completely within trimming region.
2. *OUT*: Cell completely outside trimming region.
3. *OVER*: Cell partially inside trimming region.
4. *ON*: Cell completely contains a trimming loop.
5. *ONANDOVER*: Cell partially within trimming region and fully contains a trimming loop.

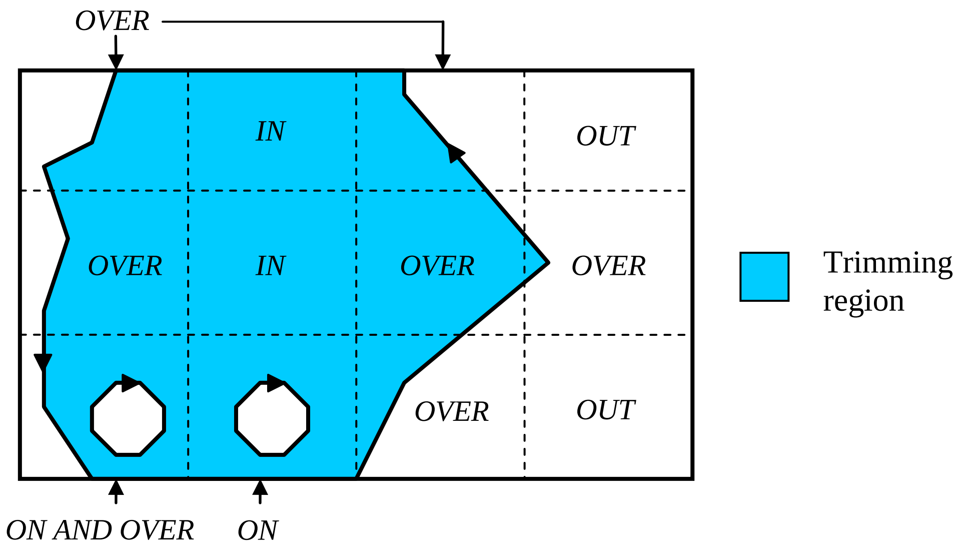


Figure 2.14 Piegl and Tiller tracing labelling convention.

Converting Boundary Edge Cells into Polygons

Once the intersections between the polygonised curve and the surface cells have been calculated, the next stage is to merge any *ON*, *OVER* and *ONANDOVER* cells, along with the trimming curves that intersect them, into polygons.

2D Delaunay Triangulation of Points

The final stage is to take the vertices defining the cells or polygons within the trimming region and triangulating them using 2D Delaunay triangulation (Section 2.2).

2.9 Sadoyan *et al*. triangulation

Sadoyan *et al*. (2006) presented a tracing scheme based triangulation algorithm.

Subdividing the NURBS based on the knot vectors

The first stage is to subdivide the base surface into a grid of cells. There are two methods presented for the determination of the *u* and *v* partition lines. The first operates as a function of the surface curvature, where the density of the grid increases in more curved areas and reduces in flatter areas. However, calculating the first and second derivatives can be computationally expensive, therefore the authors present a more efficient method, which is specific to NURBS.

Sadoyan *et al*. state that typically for NURBS surfaces and curves, the number of knot vectors to define the surface tends to grow with the curvature of the surface. Therefore the number of steps, n_u and n_v, that the parameter domain should subdivided into can be defined as

$$n_u = K \times \text{"}Number\ of\ u\ knot\ vectors\text{"} \tag{2.10}$$

$$n_v = K \times \text{"}Number\ of\ v\ knot\ vectors\text{"} \tag{2.11}$$

where *K* is an application dependent constant. For example, for visualisation applications it may vary between 1 and 10 (Sadoyan *et al*., 2005). The larger the value of *K*, the more dense the lattice will become, therefore, for applications where the level of detail is more important a higher value of *K* should be used. For applications where speed and efficiency are the priority, a smaller value of *K* can be used. The author did not provide any bounds for the value of *K*.

The NURBS trimming curves are subdivided using the same principle as for the surface.

Tracing and Triangulation

The next stage is to trace the approximated curves onto the cells, which is achieved in two steps. The first step evaluates each cell against all trimming loops to establish whether it is inside, outside or partially inside the trimming region. Cells outside the trimming region are discarded, whilst cells inside and partially inside the trimming region are stored.

The second step is to generate polygons from the stored cells. Inside cells are converted into polygons using their corners, whilst partially inside cells are converted into polygons by merging them with any trimming curves that intersect it. Once the polygons have been generated, they are triangulated using ear clipping (Section 2.1.2).

2.10 Discussion

Rockwood *et al*. (1989) present a method that converts the initial NURBS surfaces and trimming curves into Bézier form, before decomposing them into *uv* monotone regions. This requires a large amount of pre-processing and introduces extra data in the form of new boundary curves. In addition, there is no guarantee that the adjacent *uv* monotone regions will be subdivided at the same points along their boundaries edges. This can cause misalignment between triangles on adjacent patches resulting in cracks within the triangulation of a single NURBS trimmed surface (Sheng and Hirsch, 1992).

Kumar and Manocha (1995) also produce a triangulation scheme, tailored towards real-time rendering of trimmed surfaces, and use a similar approach to Rockwood *et al.*, in terms of converting the surfaces to Bézier and performing a uniform subdivision. However, they develop better bounds for uniformly triangulating the surface domain into fewer cells, and compute trimming regions without partitioning them into monotonic regions. A tracing scheme is introduced, which has found universal acceptance. A complex labelling system is used to avoid introducing extra points into the linear segments at segment/cell edge intersections. The situation where a trimming loop is completely contained within a single cell is not considered. This can occur when a design contains small holes, which are used in engineering designs as assembly features, manufacturing aides, or as means of stiffening or weight reduction.

Lukens (1996) presents an algorithm for triangulating a trimmed surface directly in its initial NURBS definition. However, the surface and curve subdivision method adopted has a tendency to over sample, producing many more triangles than is necessary. Furthermore, the method presented is purely theoretical and no implementation details or results are presented, making it difficult to assess.

Peterson (1994) creates a triangulation directly from a NURBS definition. Surfaces are adaptively subdivided, using knot refinement, based on the flatness of the control polygon (Peterson, 1994; Piegl and Tiller, 1997). The subdivided surface is then triangulated by inserting a diagonal within every cell created throughout the subdividing process. However, Peterson does not consider trimmed surfaces.

Piegl and Tiller (1998) extend Peterson's triangulation method to trimmed surfaces and adopted a simplified tracing method, similar to that of Kumar and Manocha (1995), to determine which parts of the cells are located within the trimming region. This method makes no assumptions regarding the parametric continuity of the surface. The tracing, as with Kumar and Manocha's approach, has to consider every cell, which causes many redundant checks.

Shu and Boulanger (2000) describe a tracing triangulation method which is performed within the parameter domain. The parameter domains of the surfaces and curves are adaptively subdivided using Peterson's (1994) control polygon flatness method, after which the trimming curves are then traced onto these cells to establish whether they are located outside, inside or partially inside the trimming region. However, very little detail is provided on the actual methodology used to trace the trimming curves, and degenerate cases, such as trimming curves completely contained within a cell and touch points are not considered. A touch point is defined as an end point of a trimming curve linear segment that is located on cell edge. There also seems to be no allowances for inside trimming curves which are used to describe holes within a surface.

Sadoyan *et al*. (2006) present a tracing method, similar to that of Piegl and Tiller (1998), and produce a robust uniform triangulation algorithm. Their method is generalised to operate on any trimmed surface which is parametrically defined and utilises a tracing process which can handle most degenerate cases. The polygons are then generated from these cells and triangulated using ear clipping (Section 2.1.2). No provisions are made for the situation where a trimming loop is contained within a single cell.

As with all existing structured triangulation methods, the tracing process requires that every cell, generated within the parameter domain, has to be processed with respects to every trimming loop to establish its trimming status. This can be an inefficient process for the situation especially where the trimming region forms a small proportion of the original surface. In addition, all the trimmed surface triangulation algorithms referenced above require trimming curves to have a pre-defined orientation, i.e. clockwise or anti-clockwise order.

In addition to structured triangulation schemes, unstructured triangulation generation approaches have been proposed, particularly within the field of FEA (Frey and George, 2000). In general, these algorithms require that the triangles within the final triangulation conform to some geometric criterion (i.e. edge/triangle sizes and shapes) and are as close to equilateral as possible. This is because elements that have high aspect ratios (one angle significantly smaller or larger than the other) can reduce both the solution precision and speed of its convergence (Shimada and Gossard, 1998; Shu and Boulanger, 2000). The methods operate by producing an initial triangulation of the surface using techniques such as the AFM (Lohner and Parikh, 1988; Frey and George, 2000) or Delaunay triangulation (Watson, 1981; Field, 1991). In general, the methods utilised to generate surface points, for the triangulation, are computationally intensive processes because of the geometric constraints that are imposed (Shimaga and Gossard, 1998). In addition, as the triangulation techniques adopted are incremental techniques, the integrity of each new triangle created must be checked against previous triangles, requiring a large number of calculations. Furthermore, triangulating trimmed parametric surfaces require that a triangulation of the entire surface be created first before establishing which triangles are located *IN*, *OUT* or on the *BOUNDARY* of the trimming region (Shimada and Gossard, 1998; Frey and George, 2000; Wu and Wang, 2005). For the situation where the trimmed surface forms a small portion of the underlying surface, this can produce a very large number of redundant calculations.

The Delaunay triangulation has a useful property of maximising the minimum angle of each triangle by using the circumcircle property. It is also claimed that it works towards generating a unique solution for the triangulation from a given set of points. However, this removes any flexibility for a user to manipulate the triangulation, as any changes made to the topology may cause the triangulation to no longer conform to a Delaunay triangulation, making it illegal. Furthermore, there is ambiguity in this method for the situation where more than three points lie on the circumcircle. Sugihara and Inagaki (1995) make the observation that in this situation, the polygon generated by the points on the circle will always be convex (i.e. simple polygon); therefore, a polygon triangulation method can be performed. However, this implies a unique solution will not be able to be achieved as a polygon triangulation algorithm is affected by its start point. Furthermore, the property of maximising the minimum angle will be compromised.

The AFM has the useful property of generating triangles of high quality that comply closely with desired geometric criteria. However, the resulting triangulation is largely controlled by the boundary edges of the trimming region. Therefore, any irregular triangle shapes that are formed tend to be within the centre of the trimming region, at the point where the fronts have been merged together. In addition, like the Delaunay triangulation, the triangulation algorithm does not provide much freedom in terms of topological manipulation, due to the initial geometric constraints imposed. Although a single solution may be considered as a positive attribute, the constraints imposed mean that changing the topology elements to check whether a different triangulation would yield a better approximation of the surface is not possible. This limitation prevents exploration of interesting triangulation variables, which could provide more insight into their downstream effects.

Structured triangulations provide flexibility, in terms of manipulating the topology, which could offer valuable insight into understanding the downstream effects of triangulation on flattening. Recent structured triangulation algorithms begin by subdividing the parameter domain uniformly into a grid of cells. Any cells located fully within the trimming region are triangulated into right angle triangles by adding a diagonal. The placement of the diagonal within these cells introduces an interesting degree of freedom that has not been studied before, and will be explored in detail in Chapter 5.

Another advantage of structured triangulations is that any irregular shaped triangles are restricted to the boundary edges, which results in a more uniform triangulation within the interior of the trimming region. As the boundary edges have no effect on the interior region, the entire underlying surface can be triangulated as a whole using a structured triangulation scheme. This triangulation can then be flattened into 2D and then the trimming boundaries can be mapped onto this 2D definition to recover the trimming region, if required.

Structured triangulations are also commonly used in the area of surface flattening. McCartney *et al*. (1999) and Wang *et al*. (2002) triangulate trimmed and non-trimmed surfaces using Piegl and Tiller’s (Section 2.8) approach. Cadar *et al*. (2005) worked solely with non-trimmed surfaces, which were triangulated using a uniform subdivision and right angle triangles. Part of motivation behind this thesis was prompted by Cadar *et al.*'s suggestion that triangulation could have a downstream effect on flattening. Therefore, to investigate this claim, a structured triangulation approach will be used as in previous studies (McCartney *et al*., 1999; Wang *et al*., 2002) and industry (Cader *et al*., 2005).

Little research has been reported on the downstream effects of triangulation on engineering applications. The topic of triangles with high aspect ratios has been touched upon in a variety of different areas. Shimada and Gossard (1998) noted that in FEA long thin triangles are not desired as they can cause a simulation to be less stable. Small changes in displacement of triangle vertices, during a simulation, can considerably change the shapes of triangles that have high aspect ratios, with the danger that the vertices could all become collinear. This will create degenerate triangles which can induce errors into an FEA simulation and cause the method to converge slower towards the solution. Wang *et al*. (2002) found that long thin triangles in surface flattening simulations can cause errors in the final results. This flattening error will be discussed in more detail in Section 4.4. Piegl and Tiller (1997) and Kumar *et al*. (2001) state that long thin triangles are unwanted in triangulations generated for visualisations as they can induce unnaturally looking lighting effects and creasing in a surface rendering.

It is clear that triangulation attributes can have an effect on downstream applications. However, no further triangulation attributes or their downstream effects have been reported on.

Chapter 3

NEW EFFICIENT AND ROBUST TRIANGULATION SCHEME

This chapter presents a new efficient and robust tracing scheme for triangulating trimmed parametric surfaces (Cripps and Parwana, 2011). Efficiency is gained over existing triangulation methods (Rockwood *et al.*, 1989; Kumar and Manocha, 1995, 1996; Luken, 1996; Piegl and Tiller, 1998; Shu and Boulanger, 2000; Sadoyan *et al.*, 2006) by exploiting the fact that all trimming regions are conventionally defined by closed loops of trimming curves, and adopting a uniform cell shape and enhanced data structure (Section 3.1) to minimise the number of cells processed. Key features are the efficient tracing algorithm and knowledge of orientation of the trimming curves is not required. Cells are generated by subdividing the surface parameter domain; however each cell must be represented in a specific way to achieve the tracing efficiency. Therefore this topic is introduced first followed by full details of this new triangulation algorithm. The chapter will conclude by presenting a case study that compares this new approach against the Rockwood *et al.* (1989) approach (Section 2.5), an existing trimmed surface structured triangulation scheme that is still widely used in industry.

3.1 Cell Generation and Representation

A trimmed surface is defined by a 3D parametric surface and a set of 2D trimming curves lying on the surface as shown in Section 1.5. It is essential that the curves are fully contained within the *uv* parameter domain of the surface (Section 1.5).

Generally the trimming curves have pre-defined orientations before any processing commences (Rockwood *et al*., 1989; Kumar and Manocha, 1995, 1996; Luken, 1996; Piegl and Tiller, 1998; Shu and Boulanger, 2000; Sadoyan *et al*., 2006), however, providing that each curve forms a closed loop, no further ordering information is required for the proposed algorithm.

Kumar and Manocha (1995) present a useful method for identifying trimming curves that intersect each other and propose an algorithm to split them up so that they no longer intersect. However, in general, trimming loops should not intersect, as it provides no valid geometric interpretation of a trimming region. Therefore, many schemes (Rockwood et al., 1989; Luken, 1996; Piegl and Tiller, 1998; Shu and Boulanger, 2000; Sadoyan et al., 2006), including the one presented in this chapter, assumed that no trimming curve intersects any other.

Cells, $\mathbf{C}_{i,j}$, are generated from the parameter domain by using uniform subdivision (Rockwood *et al*., 1989; Kumar and Manocha, 1995, 1996; Sadoyan *et al*., 2006). The advantage of uniform subdivision (Figure 3.1 (a)) is that it ensures rectangular cells, which are required to perform efficient tracing of the trimming loops. However, it is noted that a non-uniform subdivision of the parameter domain could be achieved by adopting a quad-tree subdivision approach (Frey and George, 2000) as shown in Figure 3.1(b). This will produce varying densities of cells whilst maintaining a regular rectangular shape, thus having a negligible impact on the tracing process and triangulation efficiency.

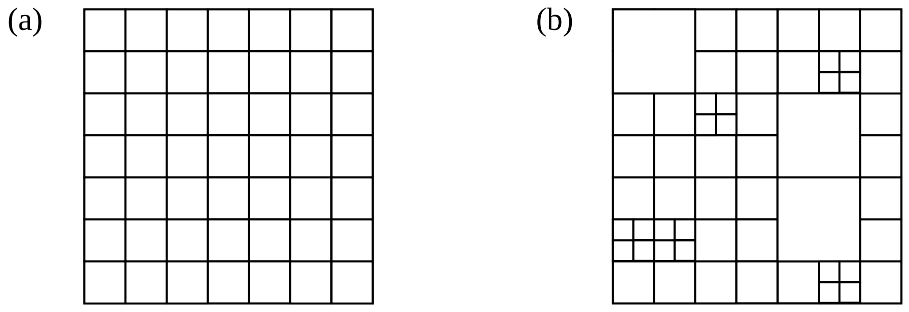


Figure 3.1 Subdivided surface parameter domains. (a) Uniform. (b) Quad-tree.

Once the subdivision has been completed, the surface parameter domain is represented as a collection of rectangular cells, with nodes and edges indexed as shown in Figure 3.2. As the cell corners are right angles, only the minimum and maximum points of each cell are needed to define it; cell edges and end point coordinates can be recovered (Figure 3.2). To simplify the tracing process (Section 3.3.3) each edge is considered to have a start and end point (half edge concept (de Berg *et al.*, 1998)), and the edges are configured in anticlockwise order. Thus, when traversing the cell edges, if the current edge location is *BOTTOM*, $\mathbf{e}_B$, the next edge would be *RIGHT*, $\mathbf{e}_R$ and the previous edge would be *LEFT*, $\mathbf{e}_L$.

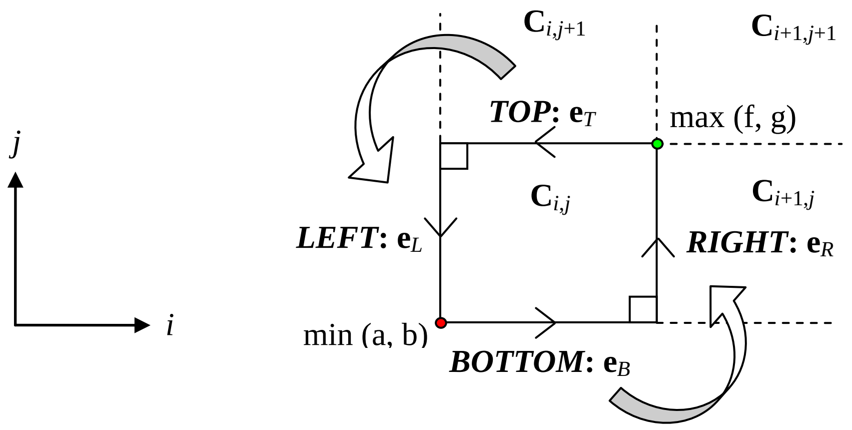
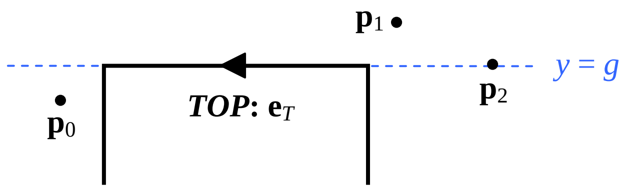

$\mathbf{C}_{i,j+1}$
$\mathbf{C}_{i+1,j+1}$
***TOP*: $\mathbf{e}_T$**
max (f, g)
j
$\mathbf{C}_{i,j}$
$\mathbf{C}_{i+1,j}$
***LEFT*: $\mathbf{e}_L$**
***RIGHT*: $\mathbf{e}_R$**
i
min (a, b)
***BOTTOM*: $\mathbf{e}_B$**

Figure 3.2 Cell, $\mathbf{C}_{i,j}$, nomenclature.

By representing the edges as half edges, each edge has a direction of motion and as the cells are rectangular, many operations can be reduced to a 1D problem. For example, to check whether a point $\mathbf{p}_i$ was to the right or collinear with the *TOP* edge, $\mathbf{e}_T$, only the y coordinate of $\mathbf{p}_i$ would need to be checked against the ray line $y = g$ (Figure 3.3).



1) $\mathbf{p}_0$ to left of $\mathbf{e}_T$
2) $\mathbf{p}_1$ to collinear $\mathbf{e}_T$
3) $\mathbf{p}_0$ to right of $\mathbf{e}_T$

Figure 3.3 Checking points against $\mathbf{e}_T$.

In addition, any points located on a particular edge can be sorted in order from front to back using only one coordinate.

3.2 Triangulation Overview

This section provides an overview of the methodology of the proposed trimmed surface triangulation algorithm. Details of the tracing process and polygon generation are given in Sections 3.3 and 3.4. Initially, the surface is subdivided into a grid of rectangular cells. The trimming loops are polygonised into piecewise linear segments and traced onto the cell structure to determine the cells that are needed for trimming loop triangulation. The tracing process conceptually involves walking around each trimming loop, moving from cell to cell, and recording all the cell/loop intersections. Only the cells which intersect the trimming curves are operated on during the tracing. The tracing process identifies all the cells along the boundary edges of the trimmed surface, which will be used, along with the cell/loop intersection data, to determine which cells are located inside the trimming region. Cells within the trimming regions can then be converted into polygons and triangulated. Once a list of polygons has been generated, the final triangulation is completed using ear clipping (Section 2.1.2). The major features of the proposed method are the tracing of the trimming loops, which is achieved efficiently because of the adopted labelling scheme, and the polygon generation. These will be described in detail in the following sections.

3.3 Tracing & Polygon Generation Details

The first stage is to trace the trimming loops onto the cell structure, whilst labelling the cells that have been intersected. The labelling and tracing processes are carried out in parallel, however, it is instructive to introduce the cell labelling first. This is followed by the description of how the intersection data is stored, which enables the polygonisation of the cells to be carried out efficiently. Tracing is described next, followed by the polygon generation algorithm.

3.3.1 Cell Labelling

All cells are initially labelled as being *OUTSIDE*, $\mathbf{C}_{i,j}^{O}$, the trimming region. Each polygonised trimming loop is selected in turn, and a search is performed to locate a cell which contains the first point of the polyline loop, within a tolerance. A point is said to be contained if it lies within the cell or on a cell edge. This trimming loop is then traversed by moving through the polyline linear segments and checking for any intersections with the cell edges. Any cell that is intersected by a trimming loop is labelled *INTERSECT*, $\mathbf{C}_{i,j}^{\dagger}$, and the intersection data is stored (Section 3.3.2). If a trimming loop is completely contained within a cell (no intersections), it is labelled *CONTAINED*, $\mathbf{C}_{i,j}^{C}$. Where a cell is intersected and has a trimming loop fully contained within it, it is labeled as *INTERSECT* and *CONTAINED*, $\mathbf{C}_{i,j}^{\dagger C}$. Figure 3.4 illustrates the tracing process.

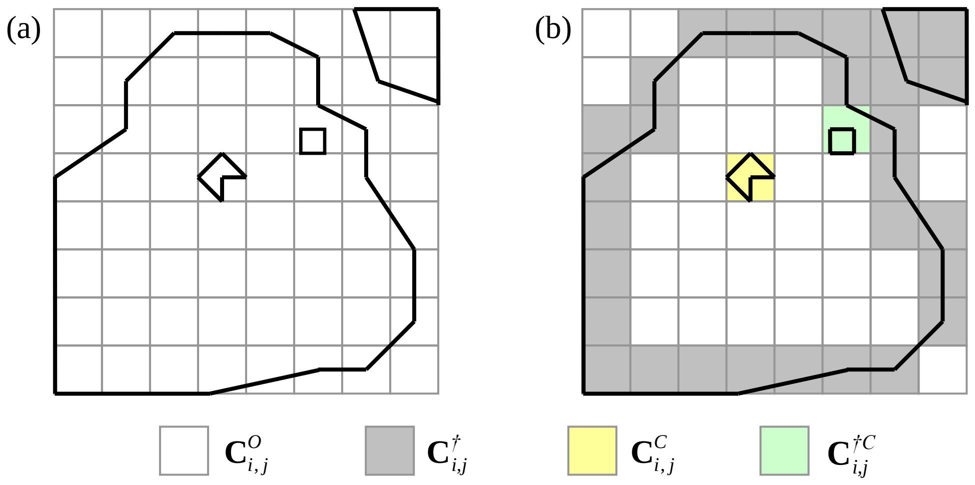


Figure 3.4 Cell labelling. (a) Pre-tracing. (b) Post-tracing.

Only the cells that intersect the trimming loops will have been processed during the tracing step. Consequently, many cells located completely within the trimming region will still be classed as $\mathbf{C}_{i,j}^{O}$, when they are in fact located *INSIDE*, $\mathbf{C}_{i,j}^{I}$. However, these cells do not need to be processed as their trimming status (i.e. whether they are $\mathbf{C}_{i,j}^{I}$ or $\mathbf{C}_{i,j}^{O}$) can be determined during the polygon generation process (Section 3.4) using the data retained from the tracing. This reduces the number cells which need to be operated on compared to existing methods (Rockwood *et al*., 1989; Kumar and Manocha, 1995, 1996; Luken, 1996; Piegl and Tiller, 1998; Shu and Boulanger, 2000; Sadoyan *et al*., 2006) which check the status of every cell against each trimming loop. For large complex surfaces with few trimmed regions, this reduction can be significant (Section 3.6.1).

3.3.2 Cell Edge Crossing Lists

During the trimming loop tracing the intersection data is stored in edge crossing lists ($\mathbf{E}_k$), which are an extension of the data structure presented by Sadoyan *et al.* (2006). When a trimming loop enters a cell through an intersection of the cell edge, it is stored in a new $\mathbf{E}_j$ as an *ENTER* point, $\boldsymbol{\varepsilon}_j$ (Figure 3.5). After $\boldsymbol{\varepsilon}_j$, the points, $\mathbf{p}_i$, of the piecewise trimming loop are then appended to $\mathbf{E}_j$ until it leaves the cell through another intersection point. This is stored as a *LEAVE* point, $\boldsymbol{\ell}_j$ (Figure 3.5). A typical $\mathbf{E}_j$ is given by: $\{\boldsymbol{\varepsilon}_j, \mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}, \mathbf{p}_n, \boldsymbol{\ell}_j\}$.

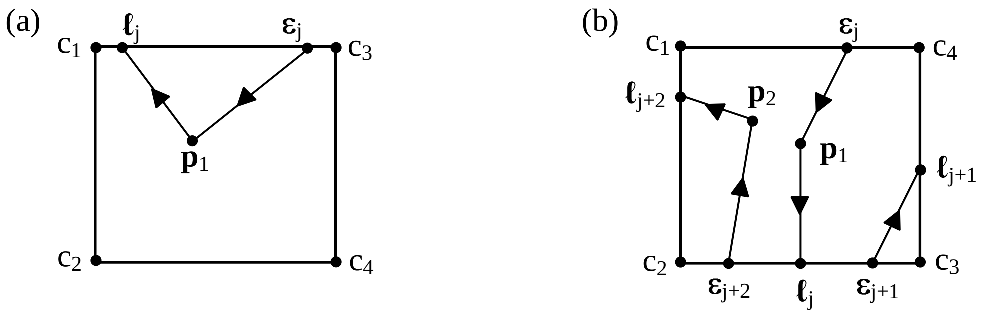


Figure 3.5 Edge crossing lists for cell $C^{\dagger}_{i,j}$. (a) Single. (b) Multiple loops.

It is worth noting that a single cell can have many $\mathbf{E}_j$'s (Figure 3.5(b)). The cell edge crossing list is extended to include a flag for each $\boldsymbol{\varepsilon}_j$ and $\boldsymbol{\ell}_j$, indicating at which edge the intersection occurred. These edge flags are used in the polygon generation process (Section 3.4).

3.3.3 Tracing Process

The tracing begins by performing a search on the cell structure to find a cell that completely contains the start point of a trimming loop. Once located, a new $\mathbf{E}_0$ is created and the first point added. The first linear segment, $\mathbf{L}_0$, of the loop is selected and checked against the cell to identify if an intersection occurs. Each linear segment $\mathbf{L}_j$ will have an associated direction: i.e. it has a start point and an end point. Determining if each polyline segment intersects a cell edge is inefficient since many will not. Therefore, an intersection identification check is introduced to avoid unnecessary calculations. If the end point of the current $\mathbf{L}_j$ is located to the right of a cell edge; an intersection has taken place. As the cell edges are represented as half edges and each cell is rectangular, this is a 1D problem. To check if the end point of $\mathbf{L}_j$ is to the right of $\mathbf{e}_T$ or $\mathbf{e}_B$, only the y values need to be compared and for $\mathbf{e}_R$ or $\mathbf{e}_L$ only the x values. To make the polygon generation process more efficient in handling degenerate cases, the intersection identification process must be performed on the cell edges in anti-clockwise order starting from $\mathbf{e}_T$ (i.e. $\mathbf{e}_T \rightarrow \mathbf{e}_L \rightarrow \mathbf{e}_\mathrm{B} \rightarrow \mathbf{e}_R$).

If no intersection is identified, then the end point of $\mathbf{L}_j$ is inserted into the current $\mathbf{E}_j$ for this cell and the next $\mathbf{L}_{j+1}$ of the trimming loop is checked. If an intersection is identified, the intersection point is calculated using standard linear intersection techniques (Schneider and Eberly, 2003) and is added to the cell $\mathbf{E}_j$ as $\boldsymbol{\ell}_j$, which terminates $\mathbf{E}_j$. The cell label is changed from $\mathbf{C}^{O}_{i,j}$ to $\mathbf{C}^{\dagger}_{i,j}$. The next cell coincident to the intersected edge is then processed. For example, if the current cell is $\mathbf{C}_{i,j}$ and the intersection occurred at $\mathbf{e}_R$, then the next cell to be traced is indexed $\mathbf{C}_{i+1,j}$; if the intersection occurred at $\mathbf{e}_T$, then the next cell is $\mathbf{C}_{i,j+1}$, etc. A new $\mathbf{E}_{j+1}$ is created and the leave point $\boldsymbol{\ell}_j$ from $\mathbf{E}_j$ is the enter point $\boldsymbol{\varepsilon}_{j+1}$ for $\mathbf{E}_{j+1}$. The enter edge flag assigned for $\boldsymbol{\varepsilon}_{j+1}$ is the opposite edge at which $\boldsymbol{\ell}_j$ occurred. For example, if the leave edge for $\mathbf{C}_{i,j}$ was $\mathbf{e}_R$, then the enter edge for $\mathbf{C}_{i+1,j}$ is $\mathbf{e}_L$.

After tracing a trimming loop, $\mathbf{E}_0$ for each new trimming loop tracing will be incomplete, i.e. it will not have an enter point, $\boldsymbol{\varepsilon}_0$, because the tracing started within the first cell. Likewise, there will be an incomplete $\mathbf{E}_n$ at the end of the tracing as it will have returned to the original cell and not have a leave point, $\boldsymbol{\ell}_n$. Therefore, $\mathbf{E}_0$ and $\mathbf{E}_n$ are merged at the end of the tracing to create a complete $\mathbf{E}_k$ for the first cell. If no intersections occur between a loop and the cell edges, $\mathbf{C}_{i,j}$ is labelled as $\mathbf{C}_{i,j}^{C}$.

3.3.4 Tracing Degenerate Cases

The intersection identification check is sufficient to handle the degenerate cases as shown in Figures 3.6 and 3.7.

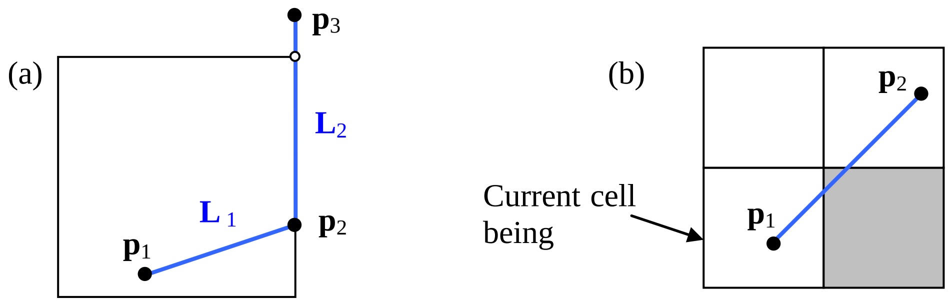


Figure 3.6 Tracing: degenerate cases.

Touch points: In Figure 3.6(a), the end point of $\mathbf{L}_1$, $\mathbf{p}_2$, is a touch point with respect to $\mathbf{e}_R$. As $\mathbf{p}_2$ is not to the right of any edge, the intersection is ignored and $\mathbf{p}_1$ is added to $\mathbf{E}_k$. $\mathbf{L}_2$ is collinear with $\mathbf{e}_R$ and since $\mathbf{p}_3$ is not located to the right of $\mathbf{e}_R$, the intersection will again be ignored with respects to this edge. However the intersection will be flagged when compared against $\mathbf{e}_T$, as the end point is located to the right of that cell edge.

Next cell identification 1: When traversing the cells, the edge checking eliminates the need to determine which cell contains the end point of the current linear segment. This also removes the degenerate case where the end point is not located in an adjacent cell. Figure 3.6(b) illustrates an example of where the linear segment intersects the current cell through $\mathbf{e}_R$. While searching for the cell which contains $\mathbf{p}_2$, the adjacent cell (shaded in Figure 3.6(b)), would be missed from the tracing since it does not contain the end point. The proposed method is able to identify this case.

Next cell identification 2: The final degenerate case is when an intersection occurs at the start or end point of a cell edge, i.e. a cell corner (Figure 3.7). A further check has to be made to establish which cell is next to be traced:

- If the intersection is at the start (end) point, then if the points of the piecewise linear segment are collinear, within a tolerance, with the previous (next) cell edge (Figure 3.7(b)), the next cell to be processed is the cell adjacent to the intersected cell edge. Otherwise, it is the cell diagonally incident to the start (end) point (Figure 3.7(a)).

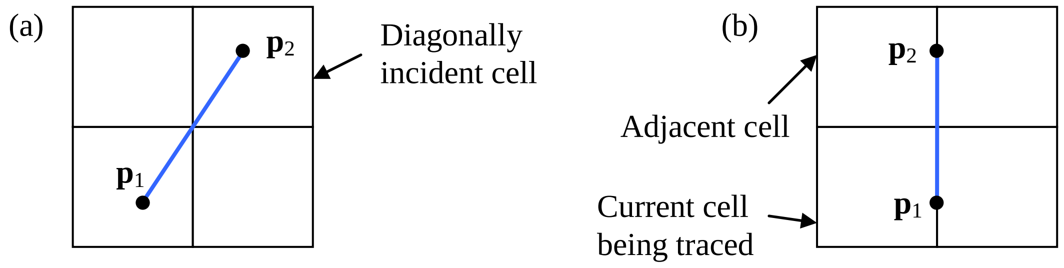


Figure 3.7 Tracing: degenerate cases.

For example, Figure 3.7(a) shows a case where $\mathbf{L}_j$ intersects $\mathbf{e}_T$ at its start point. It also intersects $\mathbf{e}_R$ at its end point, but as $\mathbf{e}_T$ is always checked first, the intersection would be calculated with respects to this edge (Section 3.3.3). The points of $\mathbf{L}_j$ are then checked to determine whether they are collinear with the previous edge. As the cell edges are orientated in anti-clockwise order, this will mean that the points will be checked against $\mathbf{e}_R$. In this case the points are not collinear, and the next cell to be traced is the cell located at the diagonal.

Figure 3.7(b) illustrates the case when points are collinear with the previous edge. Therefore, the next cell to be traced is the cell incident to the edge at which the intersection occurred.

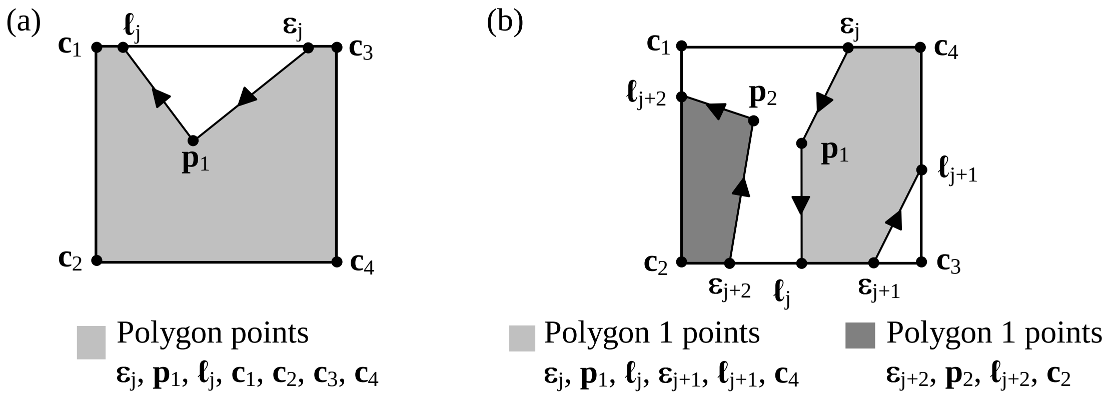
3.4 Polygon Generation

Once the tracing is complete, the next stage is to generate polygons from the appropriate cells which will then be triangulated. The method used to generate the cells is similar to that of Weiler and Atherton (1977) and Sadoyan *et al*. (2006), but has been extended to check if $\mathbf{e}_R$ of the cell is completely contained within the polygon generated from the cell.

3.4.1 Process Edge Crossing Lists

The cells are processed in rows from left to right. For every row, any cells labelled $\mathbf{C}_{i,j}^{O}$ are ignored until a $\mathbf{C}_{i,j}^{\dagger}$ or a $\mathbf{C}_{i,j}^{\dagger C}$ is reached. The polygon generation begins by taking $\mathbf{C}_{i,j}^{\dagger}$ and creating a new polygon using the points from its $\mathbf{E}_k$. The polygon is completed by traversing the cell edges in anti-clockwise order; adding the end point of each cell edge until the enter edge of the $\mathbf{E}_k$ is reached. In the situation where the $\mathbf{E}_k$ enters and leaves though the same cell edge, if $\boldsymbol{\ell}_j$ is behind $\boldsymbol{\varepsilon}_j$ on that cell edge, the polygon is complete. Otherwise, traverse the cell edges and add the cell edge end points to complete the polygon (Figure 3.8(a)).

If more than one $\mathbf{E}_k$ exists for a single cell, once the points of the first $\mathbf{E}_j$ have been appended to the polygon, an additional check must be carried out during the cell edge traversal. If any remaining $\mathbf{E}_k$'s enter in front of the current position on the current cell edge (i.e. $\boldsymbol{\varepsilon}_{j+1}$ for the polygon in Figure 3.8(b)), the points from the new $\mathbf{E}_{j+1}$ are added to the polygon. The traversal process is then continued until all the $\mathbf{E}_k$'s have been used or the original enter edge from $\mathbf{E}_j$ is reached, completing this polygon. In the situation where a polygon has been completed and some $\mathbf{E}_k$'s remain for that cell, a new polygon is created and the polygon generation process is repeated for the remaining $\mathbf{E}_k$'s.



(a)
$\mathbf{c}_1$
$\boldsymbol{\ell}_j$
$\boldsymbol{\varepsilon}_j$
$\mathbf{c}_3$
$\mathbf{p}_1$
$\mathbf{c}_2$
$\mathbf{c}_4$
Polygon points
$\boldsymbol{\varepsilon}_j, \mathbf{p}_1, \boldsymbol{\ell}_j, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4$
(b)
$\mathbf{c}_1$
$\boldsymbol{\varepsilon}_j$
$\mathbf{c}_4$
$\boldsymbol{\ell}_{j+2}$
$\mathbf{p}_2$
$\mathbf{p}_1$
$\boldsymbol{\ell}_{j+1}$
$\mathbf{c}_2$
$\mathbf{c}_3$
$\boldsymbol{\varepsilon}_{j+2}$
$\boldsymbol{\ell}_j$
$\boldsymbol{\varepsilon}_{j+1}$
Polygon 1 points
$\boldsymbol{\varepsilon}_j, \mathbf{p}_1, \boldsymbol{\ell}_j, \boldsymbol{\varepsilon}_{j+1}, \boldsymbol{\ell}_{j+1}, \mathbf{c}_4$
Polygon 1 points
$\boldsymbol{\varepsilon}_{j+2}, \mathbf{p}_2, \boldsymbol{\ell}_{j+2}, \mathbf{c}_2$

Figure 3.8 Polygons & edge crossing lists for cell $C^{\dagger}_{i,j}$. (a) Single. (b) Multiple loops.

Once $\mathbf{C}^{\dagger}_{i,j}$ is converted into a polygon, using the $\mathbf{E}_j$'s collected from tracing, they are stored for later triangulation. If a $\mathbf{C}^{\dagger C}_{i,j}$ is reached, an additional measure must be taken to establish whether the contained trimming loop(s) are holes or part of the trimming region. In general, no trimming loops should intersect. Therefore, if a point from the trimming loop is contained within any polygon generated from the cell, the loop defines a hole. Otherwise a new polygon is created using the loop points, and stored for triangulation.

3.4.2 Right Edge Checking

Since the tracing process only checks cells that intersect the trimming loops, many cells are classed as $\mathbf{C}^{O}_{i,j}$ when they are in fact within the trimming region. Therefore, in parallel with the polygon generation a check determines whether the right edge ($\mathbf{e}_R$) of the cell is completely contained within any polygon(s) generated. If $\mathbf{e}_R$ is not completely contained within the polygon (Figure 3.9(a)-(b)) any $\mathbf{C}^{O}_{i+1,j}$ cells are ignored until another intersected cell is reached. However, if $\mathbf{e}_R$ is completely within the polygon (Figure 3.9(c)) and the next cell in the row is not an intersected cell, then all $\mathbf{C}^{O}_{i+1,j}$ between the current cell and the next intersected cell in the row are definitely inside the trimming region. This is because trimming boundaries are conventionally represented as closed loops and must intersect a row of cells at least twice. Therefore, any $\mathbf{C}^{I}_{i,j}$ will always be contained between two $\mathbf{C}^{\dagger}_{i,j}$ within any given row. The status of these $\mathbf{C}^{O}_{i+1,j}$ cells is changed to $\mathbf{C}^{I}_{i+1,j}$ and they are converted into polygons, using their corners. Once an intersected cell is reached, the polygon generation and checking is repeated to establish the status of the next cells in the current row. It is observed that $\mathbf{C}^{O}_{i,j}$ and $\mathbf{C}^{I}_{i,j}$ cannot be located incident to each other in the same row.

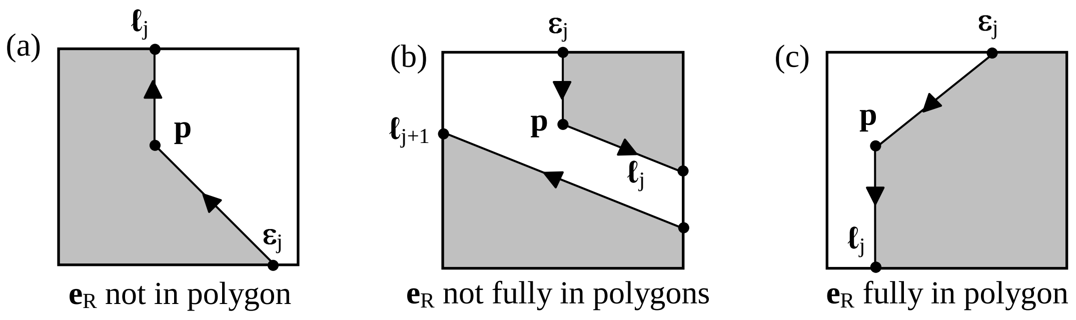

(a)
ℓ_j
p
ε_j
$\mathbf{e}_R$ not in polygon
(b)
ε_j
ℓ_{j+1}
p
ℓ_j
$\mathbf{e}_R$ not fully in polygons
(c)
ε_j
p
ℓ_j
$\mathbf{e}_R$ fully in polygon

Figure 3.9 Polygons generated from $\mathbf{C}^{\dagger}_{i,j}$ and edge crossing data.

The status of this right edge criterion is established by simply checking the enter and leave edge flags for each $\mathbf{E}_k$. For example, if the trimming loop enters the cell through $\mathbf{e}_T$ and leaves through $\mathbf{e}_B$, then the resulting polygon will fully contain $\mathbf{e}_R$ of the cell. However, if the trimming loop enters the cell through $\mathbf{e}_L$ and leaves through $\mathbf{e}_T$, then the resulting polygon will not contain $\mathbf{e}_R$. By performing similar tests for all $\mathbf{E}_k$'s, it can be established whether $\mathbf{e}_R$ is fully contained within the polygon. If multiple $\mathbf{E}_k$'s make up a single polygon and any individual $\mathbf{E}_k$ establishes that $\mathbf{e}_R$ is not fully within the polygon, then $\mathbf{e}_R$ is not in the polygon.

3.4.3 Right Edge Checking Degenerate Case

In general, edge flag checking provides enough information to determine the status of $\mathbf{e}_R$, and in turn to determine the trimming status of $\mathbf{C}_{i,j}^{O}$. The edge criterion deals with the case where $\mathbf{e}_R$ (cell labelled A in Figure 3.10) is fully within the polygon, and the next cell in the row is not within the trimming region. No extra checks are needed because the tracing method ensures that the enter edge for this $\mathbf{E}_k$ is $\mathbf{e}_B$ and the leave edge is $\mathbf{e}_T$. Checking only the edge flags means that $\mathbf{e}_R$ is not considered inside the polygon, resulting in the degenerate case being handled correctly.

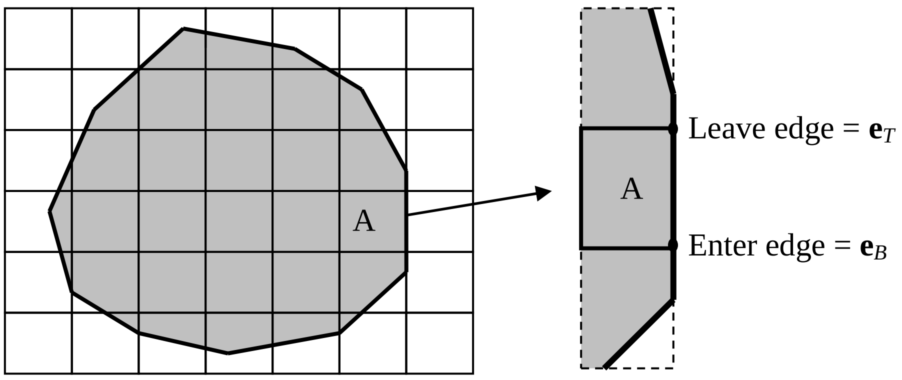


Figure 3.10 Right edge checking: degenerate case

3.4.4 Contained Cells Polygon Generation

During the polygon generation process, any $\mathbf{C}_{i,j}^{C}$ cells are handled immediately by checking their status. Since the trimming loop is fully contained within $\mathbf{C}_{i,j}^{C}$, it will have no outside influence on the status of any adjacent cells. Therefore, if the contained cell is $\mathbf{C}_{i,j}^{I}$ the trimming loop defines a hole and is stored with the polygon generated from the cell corners. Otherwise, if the cell is $\mathbf{C}_{i,j}^{O}$, the trimming loop is converted into a polygon and stored.

As the orientations of the trimming loops are not known, polygon generation and right edge checks must be carried out in a specific manner detailed in above.

3.5 Polygon Triangulation

The final task is to triangulate the list of polygons. For ease of implementation, the ear clipping algorithm (Section 2.1.2) was utilised to triangulate the polygons, which has $O(n^3)$ running time. A more efficient method, such as Seidel's (1991) randomised polygon triangulation algorithm which has a running time of $O(n\log^* n)$, would improve the efficiency of the proposed algorithm.

3.6 Case Studies

This section presents three case studies illustrating the different performance aspects of the proposed triangulation scheme (Cripps and Parwana, 2011). The trimmed surfaces used in each case study are in NUBS form, and the resulting triangulation for each test will be compared against the equivalent trimmed surface triangulation generated using the Rockwood *et al*. algorithm.

3.6.1 Efficiency

Existing methods of tracing generally process every parametric cell for each trimming loop to determine whether it is within the trimming region or not. This can result in many redundant calculations on cells to generate the final triangulation. Since the proposed method only operates on cells that intersect with trimming loops, no other cells need to be processed, improving the efficiency of the triangulation generation. The triangulation algorithm has been implemented in C++ using OpenGL and run on a standard Pentium 2 22.GHz processor. Throughout the case studies the tolerances used for point coincidence and point/edge intersections was 1.0×10^{-6}.

Trimmed surface triangulations for a range of NUBS surfaces are shown in Figure 3.11. The cells generated by subdividing the parameter domain of the original untrimmed surface have been mapped into the original untrimmed 3D surface and shown in grey.

The 'G' letter shape consists of an untrimmed spherical NUBS surface with a single trimming loop and was constructed as a simple test piece. The lamp backing, the car rear panel and the car chassis are test models provided by Delcam Plc (2010). The lamp backing consists of a non-closed curved hemispherical NUBS surface and 3 trimming loops. The outside edge and centre hole within the lamp backing are each formed by a single loop. The car rear side panel (Figure 3.11(c)) consists of a non-closed spherical NUBS and 2 trimming loops; a single exterior boundary and a hole. Finally the model toy car chassis (Figure 3.11(d)) consists of a simple plane and 12 trimming loops; an exterior boundary, a central cut-away and 10 holes.

The triangulation results for each model in terms of the number of cells that had to be processed in the triangulation construction are given in Table 3.1.

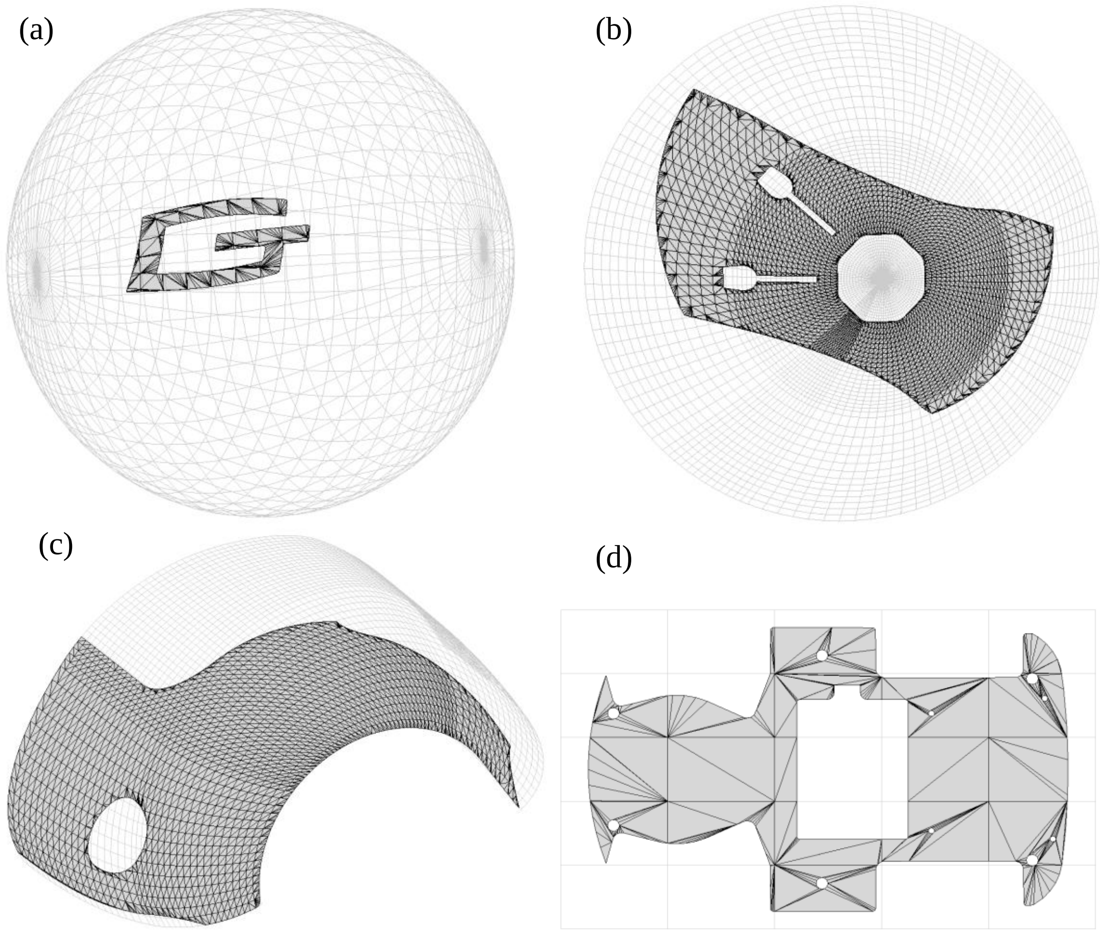


Figure 3.11 Triangulation and underlying NUBS surfaces. (a) G letter shape. (b) lamp backing. (c) car rear panel. (d) car chassis.

Surface	Loops	Cells	Cells Traced	% Traced	Tracing Time (ms)	Triangles	2D Triangulation Time (ms)
G	1	1792	48	2.68	2.87	294	3.073
Lamp	3	5500	416	7.56	49.54	5638	37.34
Car body	2	760	128	16.82	44.60	1449	32.29
Chassis	12	25	30	120.00	4.33	344	25.46

Table 3.1 Trimmed NUBS surface tracing and 2D triangulation times.

Figures 3.11(a)-(c) show that the majority of the original untrimmed surfaces are not part of the final trimmed surfaces. In addition, many cells generated from the untrimmed surface do not intersect the trimming loops. Therefore, checking a large proportion of these cells to determine their trimming status would be very inefficient. With the proposed tracing approach, the percentage of cells that were processed and traced to generate the final triangulations was 2.68% for the ’G’ shape, 7.56% for the lamp backing and 16.82% for the car rear panel. The resulting triangulations consist of 294 and 5638 and 1449 triangles for the ’G’ shape, lamp backing and car panel respectively. Even with the relatively small number of traced cells, the proposed tracing still provides sufficient information to determine the trimming status of the remaining untraced cells. This makes the operation of the modified tracing method very efficient. The car chassis (Figures 3.11(d)) is an example where the trimmed surface forms the majority of the original untrimmed surface. Larger cell sizes, i.e. less dense grids, commonly result in cells with multiple trimming loops and loops contained within a single cell. This model has 8 such trimming loops (Figures 3.11(d)). A total of 30 cells required tracing against the 12 trimming curves to construct the final triangulation comprised of 344 triangles. Table 1 also gives the time to perform the tracing and construct the 2D triangulation; showing that the processing time is dependent on the number of cells traced not the number of cells in the parametric grid. To demonstrate how this relationship can improve the efficiency of generating a triangulation, for surfaces with large cell densities, Table 3.2 shows the time taken to produce the triangulation in milliseconds for the test models which have been over sampled.

Surface	Loops	Cells	Cells Traced	% Traced	Tracing Time (ms)	Triangles	2D Triangulation Time (ms)
G	1	179200	446	0.25	10.33	3633	41.87
Lamp	3	137500	2066	1.50	78.57	100819	371.33
Car body	2	76000	1314	1.73	64.49	73215	238.30
Chassis	12	78400	1994	2.54	56.42	72124	268.74

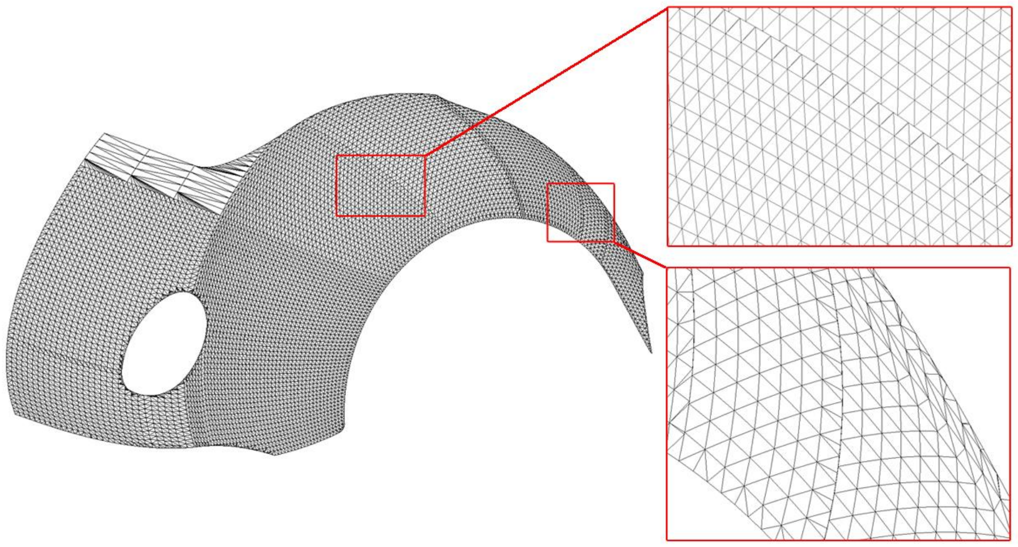
Table 3.2 Over sampled trimmed NUBS surface tracing and 2D triangulation times.

Corresponding to the models in Table 3.1, the number of cells has been increased by factors of 100, 25, 100 and 3136 for the ’G’ shape, lamp backing, car panel and chassis surfaces respectively. Despite these very large cell densities, a higher proportion of cells do not contain intersections compared to that of the original test and the percentage of cells traced by the trimming boundaries is therefore reduced. This illustrates that the relative efficiency of the proposed method is dependent on the number of cells that have to be traced, not on the density of the final triangulation. In addition, the time taken to trace only increased by factors of 4, 2, 2 and 13 for the ’G’ shape, lamp backing, car panel and chassis surfaces respectively, further highlighting the efficiency of the proposed method.

3.6.2 Crack Reduction

To illustrate the fact that the proposed method will not introduce unwanted cracks into a single NUBS surface triangulation, a trimmed NUBS surface of a car part was triangulated using the Rockwood *et al*. and the proposed tracing method. The resulting triangulations are given in Figures 3.12 and 3.13 respectively.

(a)



(b)

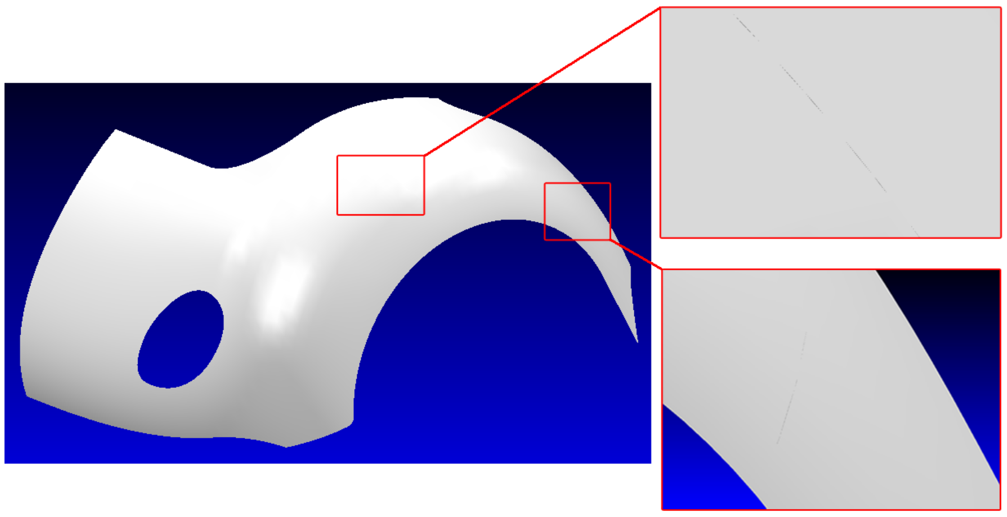


Figure 3.12 Trimmed surface triangulation using Rockwood. (b) Rendered surface.

Figure 3.12(a) clearly shows that the Rockwood *et al*. approach converts the initial NUBS surface into Bézier patches, and then further partitions these patches and trimming curves into *uv* monotone regions. These regions are then triangulated individually, creating a piecewise triangulation of the entire NUBS surface. It can be seen that the density of triangles varies between the different regions, as each is subdivided uniformly depending on the surface curvature of the region. In order to create these regions, additional trimming curves are introduced along the new patch boundaries. However, since adjacent patches can have different densities of triangles, this can result in non-coincident triangle vertices along the boundary edges on adjacent patches. Due to this, cracks can appear in the rendering of a single trimmed surface as shown in Figure 3.12(b). Whilst these cracks maybe small, Rockwood *et al*.’s method requires that highly curved surfaces be oversampled to reduce the crack sizes from appearing visually in a final rendering.

Figure 3.13(a) shows the resulting triangulation of the same NUBS surface using the proposed method.

As can be seen, there is also a change in the density of the triangles in different regions. This is because a single NUBS may be composed of many different Bézier surface patches. Since the parameter domain was subdivided uniformly across the whole NUBS surface, this has caused a change in density of *uv* spacings in the different Bézier patches. However, despite this density change, no discontinuities occur in the triangulation; the cells are always connected along the different patch boundaries hence there are no cracks in the rendering of a single trimmed surface. This has the advantage that fewer triangles can be used to triangulate highly curved surfaces, without inducing cracks into a single NUBS surface compared to a method based on converting the NUBS into a collection of Bézier patches, e.g. Rockwood *et al*.’s (Figure 3.13(b)).

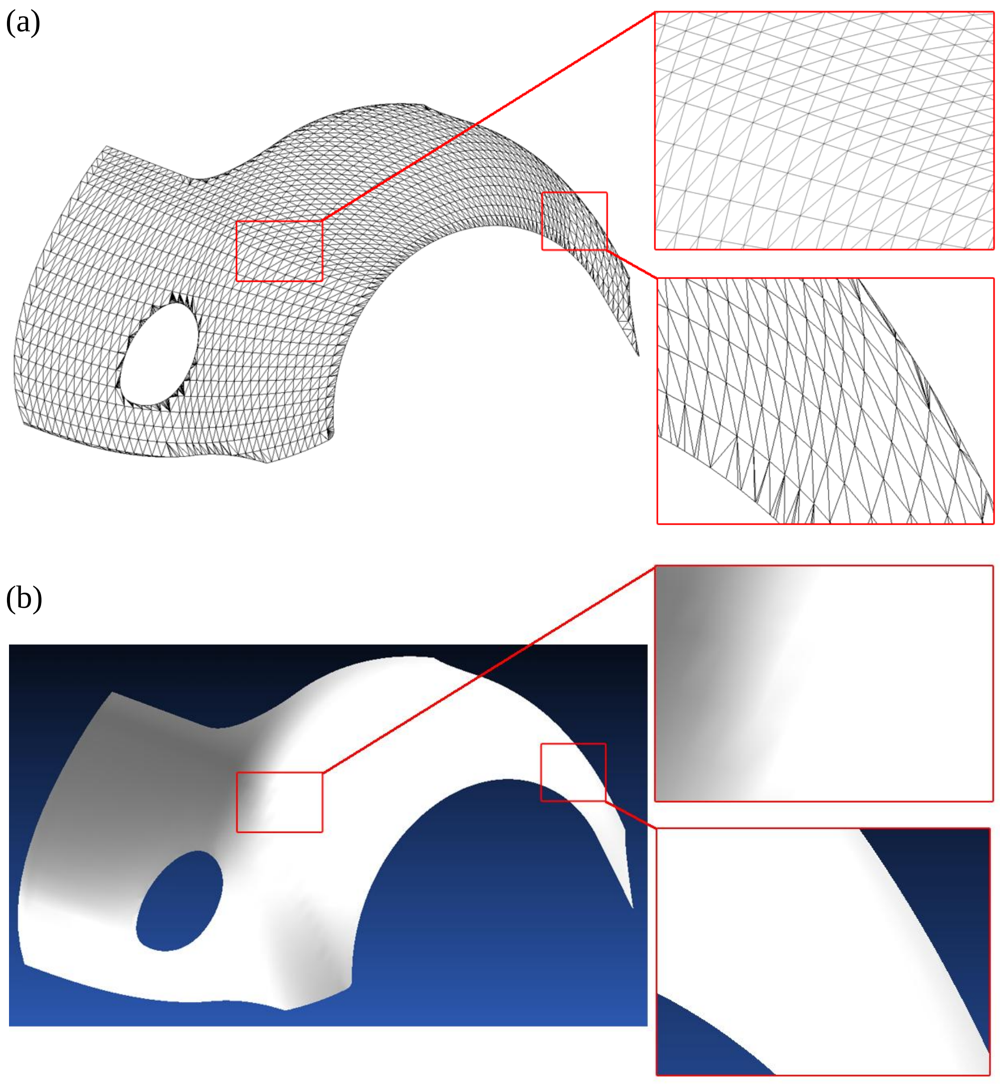


Figure 3.13 Trimmed surface triangulation using proposed method. (b) Rendered surface.

3.6.3 Skinny Triangles

The triangulation of a trimmed surface is needed for a downstream engineering application. Results from any application are dependent on the quality of the triangulation in order to produce the best possible result (Shimada and Gossard, 1998; Parwana and Cripps, 2009). The occurrence of triangles with large aspect ratios, i.e. long thin triangles, is a triangulation feature that can produce unexpected behaviour from the downstream applications. Therefore, this type of triangle should be avoided as much as possible. Nonetheless, some existing methods, e.g. Rockwood *et al*.'s, produce more triangles with large aspect ratios than is absolutely necessary. These triangulations can be refined by using a post processing remeshing technique (Shimada and Gossard, 1998; Shu and Boulanger, 2000; Wang *et al*., 2006) to make the triangles more uniform. However, the computational time taken for this step increases proportionally with the number of triangles that have to be modified. The proposed triangulation method has been optimised to efficiently generate a mesh for CAD, but generates a triangulation that is more uniform than Rockwood *et al*.'s. The percentage of triangles with high aspect ratios is also reduced because of the subdivision method and triangulation methodology utilised. Since the proposed new tracing approach generates a more uniform initial triangulation efficiently, this should aid in the computation and time taken for the remeshing process.

This is best illustrated for a simple planar trimmed surface with many trimming loops as for the toy car chassis shown in Figure 3.14.

This trimmed surface is the planar base of a toy car chassis, and consists of a single NUBS plane surface with 12 trimming loops. Rockwood *et al*.'s, method produces many triangle shapes with very high aspect ratios (Figure 3.14(a)). Since the trimming regions must be decomposed into *uv* monotone sections, these skinny triangles cannot be avoided.

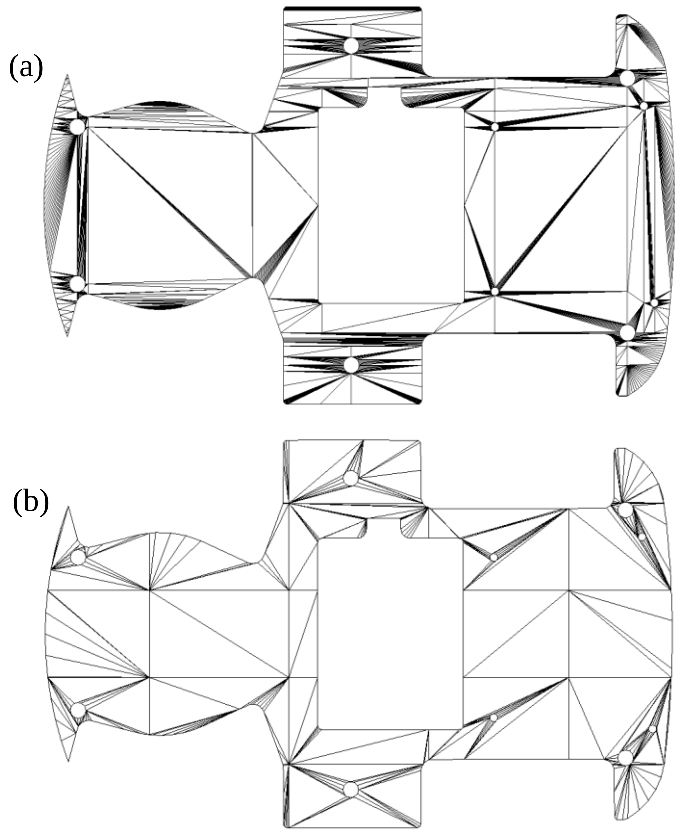


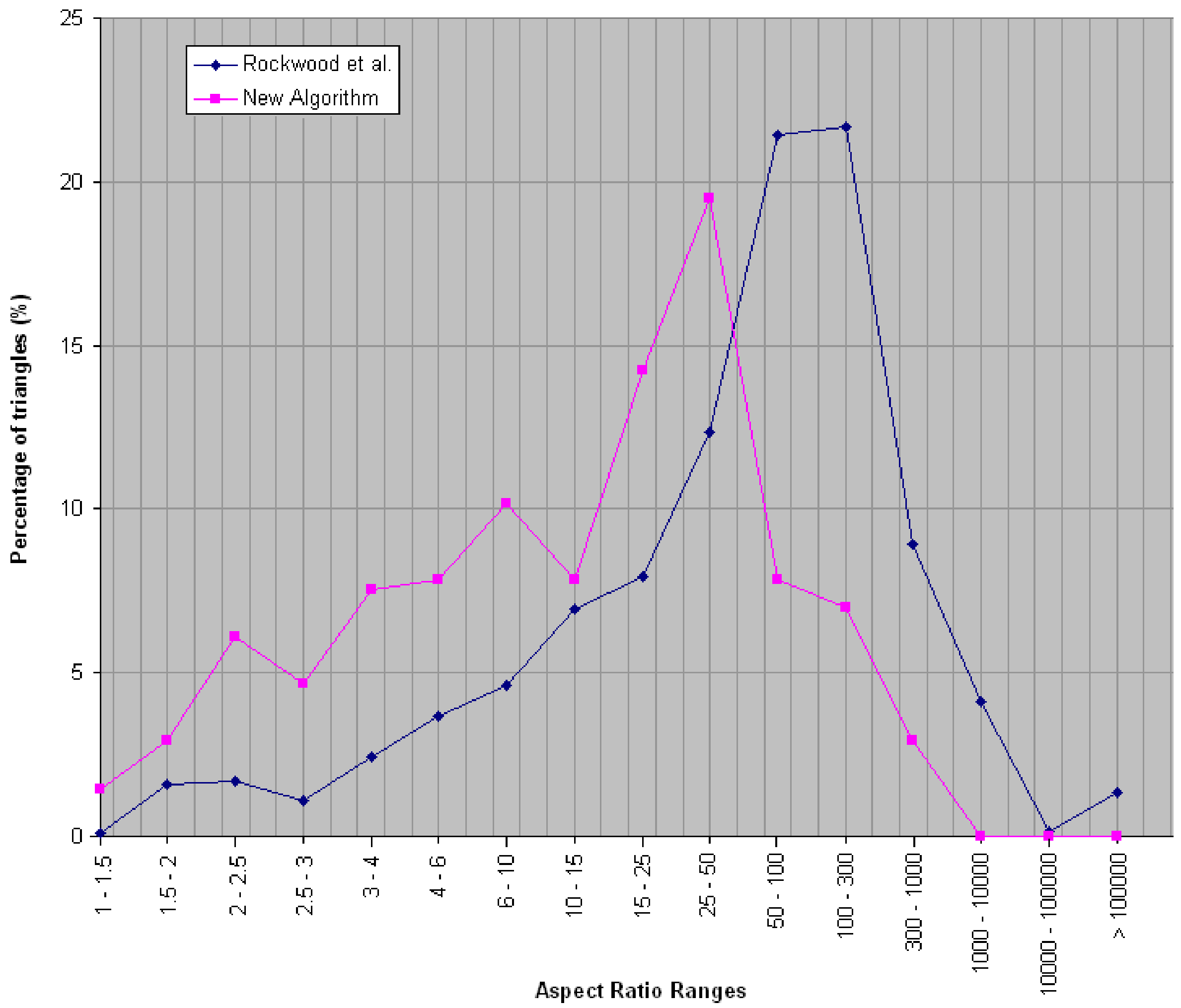
Figure 3.14 Triangulation of a trimmed surface. (a) Rockwood. (b) Proposed.

The triangulation of the same surface using the proposed method is shown in Figure 3.14(b) and the triangles are much more uniform. Any triangles of undesirable aspect ratios are confined to the trimming curve boundaries. Tables 3.3 and 3.4 show different ranges of triangle aspect ratio values and the amount of triangles, as a percentage of the total triangulation, in each aspect ratio range for the Rockwood *et al*. and new triangulation approach respectively. Graph 3.1 shows a scatter diagram of the results in these tables.

Rockwood *et al*. triangulation	
Aspect ratios	Percentage of triangles (%)
1 - 1.5	0.1
1.5 - 2	1.6
2 - 2.5	1.7
2.5 - 3	1.1
3 - 4	2.4
4 - 6	3.7
6 - 10	4.6
10 - 15	6.9
15 - 25	7.9
25 - 50	12.4
50 - 100	21.4
100 - 300	21.7
300 - 1000	8.9
1000 - 10000	4.1
10000 - 100000	0.2
> 100000	1.3

New triangulation approach	
Aspect ratios	Percentage of triangles (%)
1 - 1.5	1.5
1.5 - 2	2.9
2 - 2.5	6.1
2.5 - 3	4.7
3 - 4	7.6
4 - 6	7.8
6 - 10	10.2
10 - 15	7.8
15 - 25	14.2
25 - 50	19.5
50 - 100	7.8
100 - 300	7.0
300 - 1000	2.9
1000 - 10000	0
10000 - 100000	0
> 100000	0

Tables 3.3 and 3.4 Aspects ratios of triangles within the triangulation.



Graph 3.1 Aspects ratios of triangles within the triangulation.

From the tables and graph it can be seen that the majority of triangles in the new approach come under a smaller aspect ratio range than the Rockwood *et al*. approach. This results in triangles that are more uniform and reduces the number of long thin triangles. Furthermore, the minimum and maximum aspect ratios for the proposed triangulation approach were 1.19 and 870.93 respectively. However, the Rockwood *et al*. approach had a minimum aspect ratio of 1.50 and a maximum ratio of ∞ (i.e. triangle vertices were all collinear) creating a degenerate triangle. These long thin triangles (Figure 4.13(a)) are created because the surface has many small curves and loops which have forced the Rockwood *et al*. method to generate unnecessary *uv*-monotone regions. Furthermore, the subdivision methods adopted by Rockwood *et al*. tend to over sample the number of points generated in an attempt to reduce the occurrence of cracks along the boundary edges when the regions are stitched back together. This, compounded with the tiling and coving method, used to triangulate the boundary edges, produces many degenerate triangles as has been noted by Rockwood *et al*. (1989).

Chapter 4

SURFACE FLATTENING

The method of surface flattening, sometimes referred to as unwrapping, is used to map a 3D surface onto a 2D plane (Figure 4.1), and is applicable to industries that manufacture shoes, clothing and decorations for textiles and ceramics. These fields of work all involve making an item from a 2D material, which will ultimately be placed onto a 3D model, such as a tea pot or human body part. In conventional practice, a 2D pattern is designed manually by a field expert, which is then used to create the item from the 2D material. Once completed, it is placed onto the 3D model, after which alterations can be made to produce a better fit. Designing an initial 2D pattern is a specialised craft and can be a complicated task. However, use of flattening simulation allows a designer to create the final 3D model, which can then be flattened to produce a 2D pattern. It has been shown that this approach can produce 2D patterns that have a better initial fit as well as decreased production lead-times (Wang *et al*., 2002; Cader *et al*., 2005).

A widely used method for flattening 3D surfaces is to first approximate the geometry as a set of 3D triangles, then map the 3D triangular facets into 2D as presented by McCartney *et al*.

(1999) and later modified by Cader *et al*. (2005). The triangles are sequentially flattened into 2D by starting from a seed triangle and then operating on the triangles that share an edge with any flattened triangle.

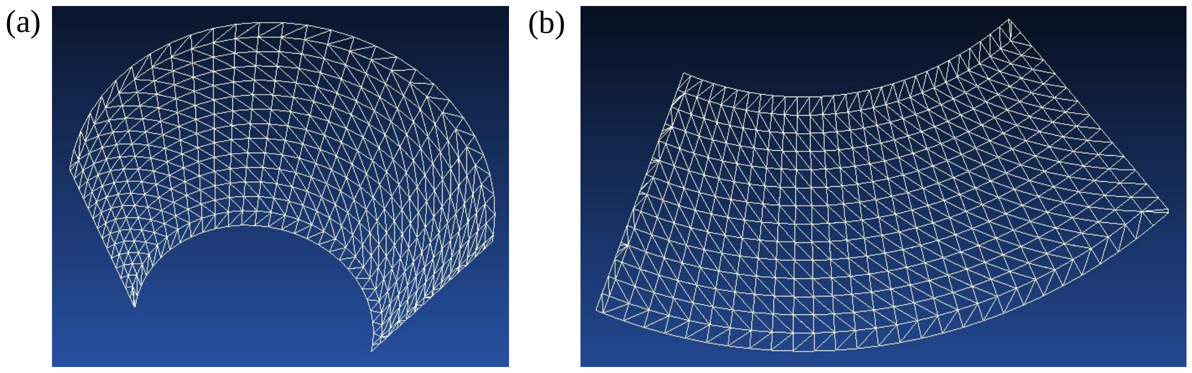


Figure 4.1 Cone section. (a) 3D triangulation. (b) unwrapped triangulation.

As the original 3D triangulation will be mapped into the 2D plane of the seed triangle, some distortion may occur within the flattened triangulation (McCartney *et al*., 1999; Parwana and Cripps, 2009), which is dependent on the surface curvature. If a surface is developable (Section 1.4.4), then there will be no distortion when it is flattened, otherwise, there will be some distortion depending on the magnitude of the curvature (McCartney *et al*., 1999).

This chapter will present the details of both the McCartney *et al*. (1999) and Cader *et al*. (2005) flattening processes. It will also present known surface and algorithmic characteristics that can have an effect on the final flattening result.

4.1 McCartney *et al*. Algorithm

One of the most widely used algorithms for flattening is McCartney *et al*.'s method (McCartney *et al*., 1999). This method begins by triangulating the surface using Piegl and

Tiller's (1998) triangulation approach, which generates a set of triangles, **T**, consisting of mainly right angle triangles. An initial flattening is then produced by constraining the edge lengths of each triangle in 2D to be the same as the corresponding 3D triangle.

4.1.1 Circle Intersection Algorithm

The first triangle to be flattened is chosen from the triangulation and is known as the seed triangle, $\mathbf{t}_s$, that has the 3D vertices, ($\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$), (Figure 4.2(a)) and edge lengths

$$l_0 = \|\mathbf{p}_1 - \mathbf{p}_0\|, \quad l_1 = \|\mathbf{p}_2 - \mathbf{p}_1\|, \quad l_2 = \|\mathbf{p}_0 - \mathbf{p}_2\| \tag{4.1}$$

The first two vertices are placed into 2D using the coordinates $\mathbf{p}'_0 = (0,0)$ and $\mathbf{p}'_1 = (l_0, 0)$. The next stage is to create two 2D circles $\mathbf{C}_0$ and $\mathbf{C}_1$, which have the centre points $\mathbf{p}'_0$, and $\mathbf{p}'_1$, as well as radii l_0, and l_1 respectively. The position of the third vertex $\mathbf{p}'_2$ in 2D is determined by the intersection point between $\mathbf{C}_1$ and $\mathbf{C}_2$ (Schneider and Eberly, 2003) (Figure 4.2(b)).

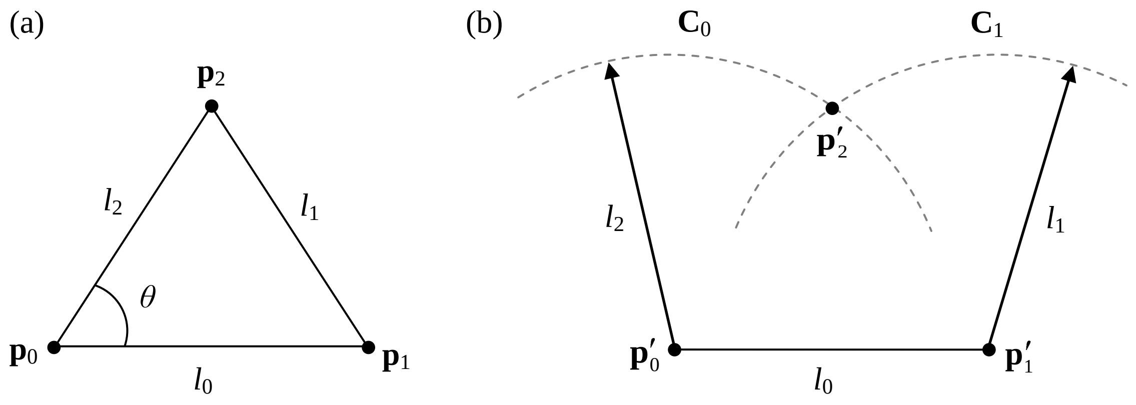


Figure 4.2 (a) 3D triangle. (b) 2D flattening of 3D triangle.

Once the seed is flattened, any triangle in **T** that shares an edge with $\mathbf{t}_s$ may be flattened next. As the next triangle will share an edge with $\mathbf{t}_s$, two of its points will already have been flattened. Therefore, only the third vertex needs to be calculated and this is done by using the circle intersection method.

Instances where all three vertices of a triangle have already been flattened will occur during the process, which will mean that one vertex in 3D may have more than one position in 2D.

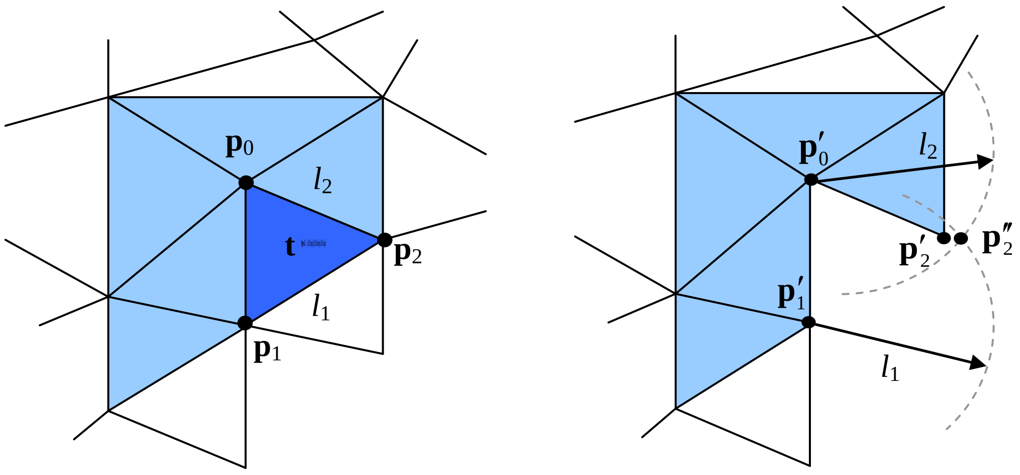


Figure 4.3 (a) 3D triangulation. (b) Constrained flattening.

This can be illustrated in Figure 4.3(a) where the 3D triangle, **t**, is next to be flattened, however, all of its vertices have already been mapped into 2D. Therefore, when mapping **t** into 2D by determining a new position for $\mathbf{p}_2$, if the surface is non-developable, the circles, $\mathbf{C}_0$ and $\mathbf{C}_1$, may intersect at a different point than that of $\mathbf{p}'_2$ (Figure 4.3(b)). When $\mathbf{p}''_2$ is connected to $\mathbf{p}'_0$ and $\mathbf{p}'_1$ this will cause tearing or overlapping (Figure 4.4). If the surface is developable, the multiple 2D positions will all be the same, within tolerance, and therefore no distortion should occur.

(a) (b)

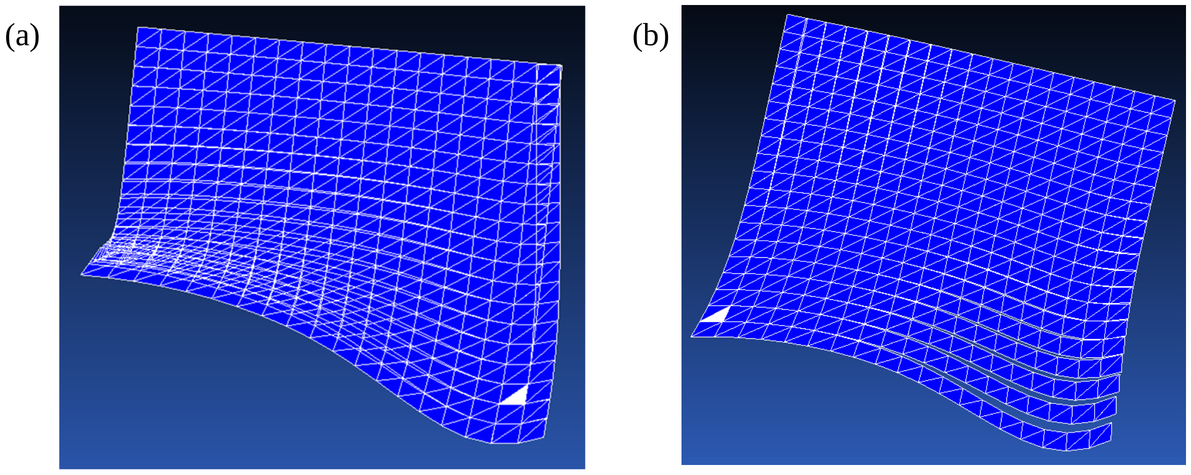


Figure 4.4 Examples of (a) overlapping and (b) tearing in a 2D flattening.

4.1.2 Energy Release Method

To remove any distortion which can arise from flattening surfaces with non-zero Gaussian curvature, McCartney *et al*. use an energy release method to reduce strain energy from the flattened triangulation, thus attempting to reconnect the topology. The strain energy associated with an edge, $\mathbf{e} = (\mathbf{p}_0, \mathbf{p}_1)$, is given by

$$f_{energy}(\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}'_0,\mathbf{p}'_1) = 0.5\frac{(\mathbf{p}'_0\mathbf{p}'_1 - \mathbf{p}_0\mathbf{p}_1)^2}{\mathbf{p}_0\mathbf{p}_1} \tag{4.2}$$

where $\mathbf{p}_0$ and $\mathbf{p}_1$ are the 3D vertices of the edge and $\mathbf{p}'_0$ and $\mathbf{p}'_1$ are the flattened 2D vertices of the edge. The energy release method takes each vertex, $\mathbf{p}'_i$, in the flattened triangulation and attempts to move it an incremental distance of Δd in four orthogonal directions $\mathbf{p}'_{ia}$, $\mathbf{p}'_{ib}$, $\mathbf{p}'_{ic}$, $\mathbf{p}'_{id}$ as shown in Figure 4.5. McCartney do not state how to determine Δd.

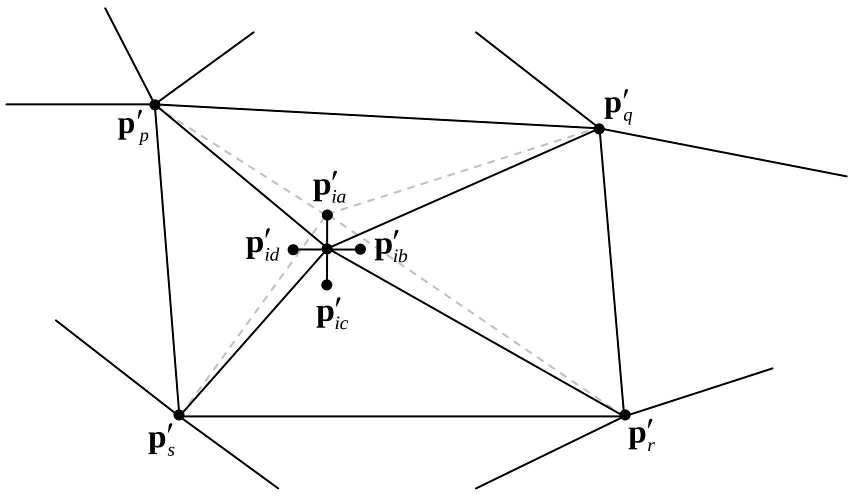


Figure 4.5 Orthogonal trial movements for energy release.

Suppose that the 3D equivalent vertex of $\mathbf{p}'_i$, in the original 3D triangulation was connected to the vertices $\mathbf{p}_p$, $\mathbf{p}_q$, $\mathbf{p}_r$, $\mathbf{p}_s$, then each vertex would have an associated energy value of E_i, which is the sum of the strain energy for all connected edges, i.e.

$$\begin{gathered} E_i = f_{energy}(\mathbf{p}_i, \mathbf{p}_p, \mathbf{p}'_i, \mathbf{p}'_p) + f_{energy}(\mathbf{p}_i, \mathbf{p}_q, \mathbf{p}'_i, \mathbf{p}'_q) + \\ f_{energy}(\mathbf{p}_i, \mathbf{p}_r, \mathbf{p}'_i, \mathbf{p}'_r) + f_{energy}(\mathbf{p}_i, \mathbf{p}_s, \mathbf{p}'_i, \mathbf{p}'_s) \end{gathered} \tag{4.3}$$

Each orthogonal test move is then tried on vertex $\mathbf{p}'_i$ and the total new strain energy value calculated and stored. For example, moving $\mathbf{p}'_i$ to $\mathbf{p}'_{ia}$ (Figure 4.5) would yield the strain energy value E_{ia} given by:

$$\begin{gathered} E_{ia} = f_{energy}(\mathbf{p}_i, \mathbf{p}_p, \mathbf{p}'_{ia}, \mathbf{p}'_p) + f_{energy}(\mathbf{p}_i, \mathbf{p}_q, \mathbf{p}'_{ia}, \mathbf{p}'_q) + \\ f_{energy}(\mathbf{p}_i, \mathbf{p}_r, \mathbf{p}'_{ia}, \mathbf{p}'_r) + f_{energy}(\mathbf{p}_i, \mathbf{p}_s, \mathbf{p}'_{ia}, \mathbf{p}'_s) \end{gathered} \tag{4.4}$$

The energy values E_{ib}, E_{ic}, E_{id} for the test moves $\mathbf{p}'_{ib}$, $\mathbf{p}'_{ic}$, $\mathbf{p}'_{id}$ respectively, are calculated in a similar way. The maximum energy reduction, ΔE_i, caused by an orthogonal movement is then calculated by $\Delta E_i = \max\{(E_i - E_{ia}), (E_i - E_{ib}), (E_i - E_{ic}), (E_i - E_{id})\}$. If ΔE_i is greater than a user defined energy threshold, E_t, then the orthogonal movement associated with ΔE_i

will be applied to vertex $\mathbf{p}_i'$. A value less than or equal to E_t, within tolerance, will mean that $\mathbf{p}_i'$ is not considered for movement.

The energy release process is then applied to all vertices connected to $\mathbf{p}_i'$, eventually removing strain energy for every vertex within the flattened topology. The process is terminated when no further reductions of strain energy are greater than E_t. This can mean that a single vertex can be tested many times making this a very computational intensive process. This can be illustrated by a result presented by McCartney *et al.* (1999), in which they flattened a section of a torus with the dimensions shown in Figure 4.6.

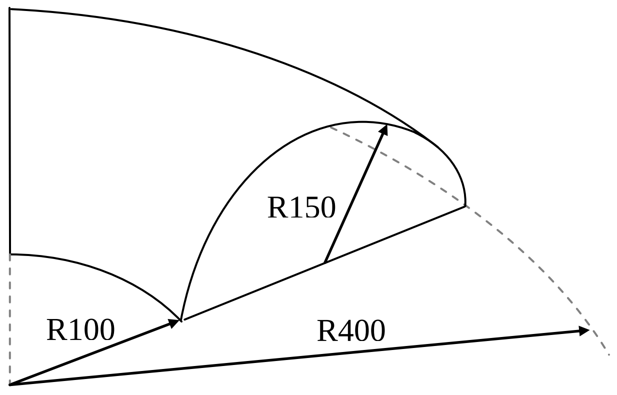


Figure 4.6 Part of a torus.

The proportion of the torus used and the number of triangles was not provided in the results. However, for an energy threshold of E_t = 1e^{-4}, the energy release process took over 6 minutes to complete. It is stated that a value of E_t = 3e^{-4}, is a good compromise between accuracy and processing time, however, a simulation with this value still took over 3 minutes (McCartney *et al.*, 1999).

A way to resolve these long processing times would be to iteratively change the topology as distortion occurs in the system. However, due to the circle intersection method employed, runtime modifications of the flattened topology are not possible and have to be postponed until the end. This is because flattened vertices act as centre points for circles and moving a vertex will essentially move a possible centre of a circle. Over time, this movement can accumulate so that the distance between the centre points is large enough to force the circles to no longer intersect one another.

4.2 Cader *et al.*'s Flattening Method

Cadar *et al.* (2005) proposed an improvement to McCartney *et al.*'s algorithm, which is essentially a modification of the McCartney *et al.* flattening scheme. The modified algorithm utilises a cosine method to determine the position of the triangle vertices in 2D, which is equivalent to McCartney *et al.*'s intersection of circles approach.

4.2.1 Cosine Algorithm

In the case of the seed triangle, $\mathbf{t}_s = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)$, the first two vertices would be flattened using the same method as McCartney *et al.* The edge lengths, given by (4.1), are then used to calculate the subtended angle θ at $\mathbf{p}_0$ (Figure 4.3(a)) using the cosine rule

$$\theta = \cos^{-1}\left(\frac{l_0^2 + l_2^2 - l_1^2}{2l_0 l_2}\right)$$

Therefore, the coordinates of the three flattened vertices $(\mathbf{p}_0', \mathbf{p}_1', \mathbf{p}_2')$ which preserve the 3D edge lengths of the original triangle, are given as

$$\mathbf{p}_0' = (0,0), \quad \mathbf{p}_1' = (l_0, 0), \quad \mathbf{p}_2' = (l_2 \cos\theta, l_2 \sin\theta)$$

As in McCartney's algorithm, once a seed has been selected, this method continues to flatten triangles incident to previously unwrapped triangles. However, flattening these remaining triangles is a little more complicated compared to flattening the seed. The process is illustrated with an example.

Figure 4.7(a) shows triangles $\mathbf{t}_0$ and $\mathbf{t}_1$. $\mathbf{t}_0$ has already been mapped into 2D, $\mathbf{t}'_0$, (Figure 4.7(b)), so $\mathbf{t}_1$ is the next triangle to be processed and shares an edge with $\mathbf{t}_0$ and vertex $\mathbf{p}_3$ of $\mathbf{t}_1$ is the only vertex position to be determined in 2D. The edge lengths and angle θ are calculated ensuring that the angle is the one which is incident to the start point of the half edge ($\mathbf{p}_s$) which is incident to $\mathbf{t}_1$ (Figure 4.7(c)).

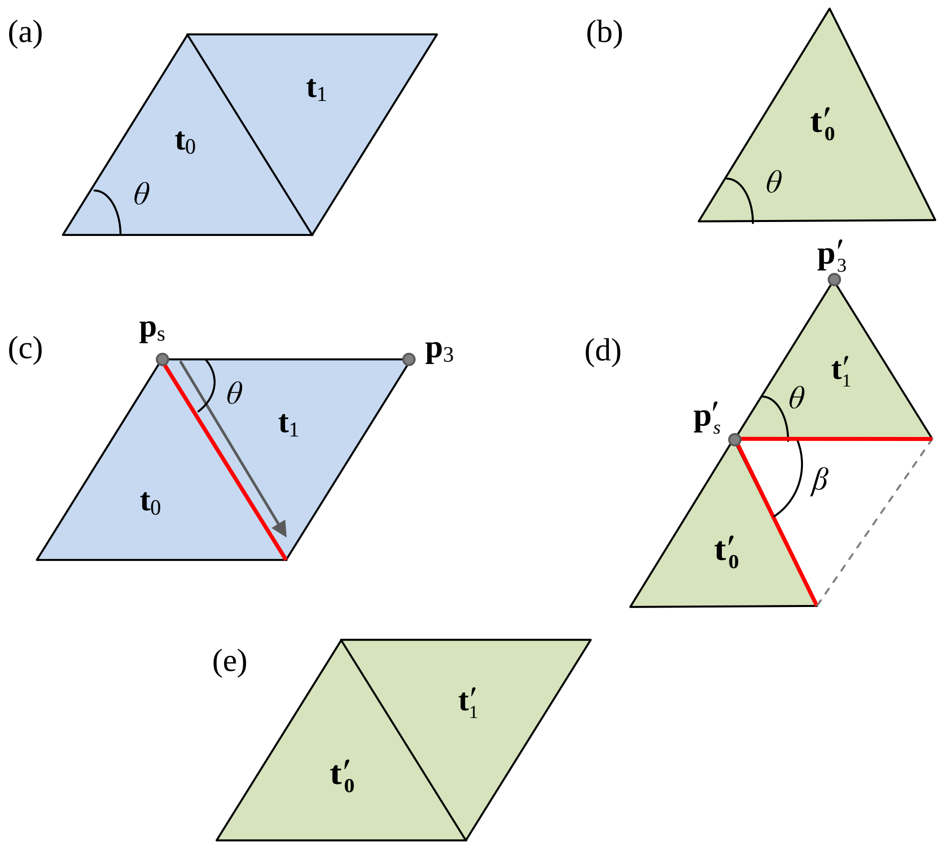


Figure 4.7 Cosine flattening process. Blue triangles are 3D and green 2D. (a) 3D triangulation. (b) Seed flattened. (c) Identifying $\mathbf{p}_s$ and which vertex to flatten.

Once θ has been calculated, the new positions of the two vertices are calculated relative to the $\mathbf{p}_s$ of the shared half edge as shown in Figure 4.7(d).

As can be seen in Figure 4.7(d), the final stage is to rotate the triangle into the correct position. This is done by calculating angle β, again using the cosine rule, and then rotating $\mathbf{p}_2$ though β about the start point of the incident half edge (Figure 4.7(e)). This method is continued until all triangles have been flattened.

This process produces a result which is equivalent to the circle intersection method (McCartney *et al*., 1999), but has the advantage that it allows for runtime modifications of the flattened triangulation, which eliminates any tearing or overlapping as they occur. Therefore, in the situation where a duplicate 2D position for an existing flattened vertex has to be calculated, an unbiased average of the two different positions is taken to create a single position. It is again worth noting that if the surface is developable, the position of the two vertices will be the same, within tolerance.

4.2.2 Mapping Order

Cader *et al*. (2005) also present a method for pre-determining the order in which a 3D triangulation will be flattened into 2D. This is opposed to McCartney *et al*.'s method which determines the next triangle to be flattened on the fly and in no logical order. The seed triangle is denoted as level 0 and will be the first triangle to be flattened. Any triangle which shares an edge with the seed triangle will be denoted level 1, and any triangle which shares an edge will a triangle from level 1 will be denoted as level 2, etc.

The order in which triangles are flattened can change the final flattening result and is an ambiguity currently present in some methods. Therefore, given a 3D triangulation and a seed triangle, two implementations of the McCartney *et al*. method could easily yield completely different flattening results, due to the ambiguity in the mapping order. Using a consistent mapping order can reduce this ambiguity, resulting in more consistent results when flattening a surface. In addition, due to the circular nature of the mapping order, it flattens the triangulation evenly around the seed, which ensures that any distortion is distributed evenly throughout the entire surface.

4.3 Accuracy Measurements

The accuracy of a flattening simulation can be assessed using differences in area and shape as proposed by Wang *et al*. (2002). The area difference measures the change in surface area pre- and post-flattening. The relative area difference is given by

$$\delta A = \frac{A^o - A^f}{A^o} \tag{4.5}$$

where A^o is the original area of the 3D surface and A^f is the area of the 2D flattened surface. Since analytical forms of the flattened surface cannot be determined, the areas are approximated by summing the triangles areas within each triangulation, i.e.

$$A^o = \sum_{i=0}^{n-1} \Delta_i^o \quad \text{and} \quad A^f = \sum_{i=0}^{n-1} \Delta_i^f \tag{4.6}$$

where Δ_i^o is the area of the ith triangle from the 3D triangulation consisting of n triangles and Δ_i^f is the are of the ith flattened triangle.

The shape difference measures the change in the surface curve lengths pre- and post-flattening, where the curve length is defined as the length of an arbitrary iso-parameter line on the surface. The final relative edge length difference is given by

$$\delta S = \frac{L^o - L^f}{L^o} \tag{4.7}$$

Where L^o is the actual curve length of the curve segment on the original surface and L^f is the corresponding curve length in the flattened model. Again, analytical forms of the curve length cannot be determined for the flattened model, so L^o and L^f are approximated by summing the length of each edge in the corresponding triangulation, i.e.

$$L^o = \sum_{j=0}^{m-1} E_j^o \quad \text{and} \quad L^f = \sum_{j=0}^{m-1} E_j^f \tag{4.8}$$

where E_j^o is the edge length of the jth edge in the 3D triangulation consisting of m edges and E_j^f is the edge length of the jth edge in the flattened triangulation.

4.3.1 Improved Accuracy Flattening Measures

Although the measures presented in (4.5) and (4.7) are used widely, it is noted that they can give misleading results, which can portray an incorrect result. This is because it is very possible to induce large amounts of area distortion within a flattening of a triangulation, yet achieve $\delta A = 0$. For example, during the flattening of 10 triangles, 5 triangles could increase in area by a half and 5 triangles could reduce in area by a half after flattening. Summing these area changes, the total area after flattening would be the same as the original 3D area. This would imply that no area distortion took place, yet clearly every triangle induced distortion. Therefore, this measure would give a poor reflection of the actual distortion induced. This argument is analogues for the shape change.

Therefore, two new additional measures are defined to reflect the measure of distortion after flattening more accurately. The first is defined as the absolute area change given by

$$\Delta A = \sum_{i=0}^{n-1} \left| \Delta_i^o - \Delta_i^f \right| \quad (4.9)$$

The second is defined as the absolute shape change given by

$$\Delta S = \sum_{j=0}^{m-1} \left| E_j^o - E_j^f \right| \quad (4.10)$$

In each of these measures, the absolute difference in area (shape) per triangle (edge) is measured and summed together, meaning that all distortions will be accounted for in this measurement.

4.4 Discussion

McCartney *et al.* (1999) present a triangular based approach for flattening a 3D triangulation into the 2D plane of the seed triangle, by constraining the edge lengths of the 3D triangles. Due to the circle intersection procedure adopted, if the surface is non-developable, the initial flattening of the surface may induce tearing or overlapping of triangular facets. It is also stated that if the surface is developable, then the triangulation of that surface will also flatten without distortion. To remove any tearing or overlapping, an iterative energy release method is used to reconnect the triangulation. No definitive mapping order for the triangulation flattening is provided; meaning that different implementations of this technique can essentially flatten the triangles in a different order. This can be a problem, as noted by Cader *et al.* (2005), since the order in which the triangles are flattened into 2D can effect the resulting flattening.

The energy release method adopted by McCartney *et al.* is a very computationally intensive task which has to test many different solutions for a new point position before settling on the final answer. The large computational time is due to the fact that the energy release method must be used as a post-processing step on the entire flattened structure. This is imposed on the algorithm because runtime modification of vertex positions can cause the flattening to fail in situations where the cumulative movement of the points becomes so much that the circles no longer intersect.

In practice, surface flattening is used on developable or non-developable surfaces with very small curvature (almost developable); therefore, Cader *et al.* (2005) presented a flattening method that constrains the edges lengths of triangles and produces a result equivalent to McCartney *et al.* By utilising a cosine method rather than an intersection of circles, Cader *et al.* were able to incorporate runtime modifications into their algorithm. Therefore, multiple 2D positions for a single 3D vertex can be removed immediately by taking the unbiased average of the points.

4.4.1 Highly Curved Surfaces Problem

As the method of Cadar *et al.* proposed an improvement to McCartney *et al.*'s algorithm, it is adopted as a suitable scheme with which to demonstrate the effects of triangulation. However, as the method has been optimised for almost developable surfaces, problems can arise when working with highly curved surfaces. For these types of surfaces, the averaging process can induce large changes to the shape of flattened triangles and cause them to flip from anti-clockwise to clockwise as illustrated in Figure 4.8. Figure 4.8(a) shows two incident triangles $\mathbf{t}'_0 = (\mathbf{p}'_0, \mathbf{p}'_1, \mathbf{p}'_2)$ and $\mathbf{t}'_1 = (\mathbf{p}'_2, \mathbf{p}'_3, \mathbf{p}'_4)$ which have been flattened into 2D. Vertices $\mathbf{p}'_0$ and $\mathbf{p}'_4$ represent the same vertex in 3D and need to averaged to establish a single position in 2D to reconnect the triangles. Therefore, the updated triangle vertices become $\mathbf{t}'_0 = (\mathbf{p}'_a, \mathbf{p}'_1, \mathbf{p}'_2)$ and $\mathbf{t}'_1 = (\mathbf{p}'_2, \mathbf{p}'_3, \mathbf{p}'_a)$, however, this has caused $\mathbf{t}'_0$ to change from anti-clockwise to clockwise (Figure 4.8(b)).

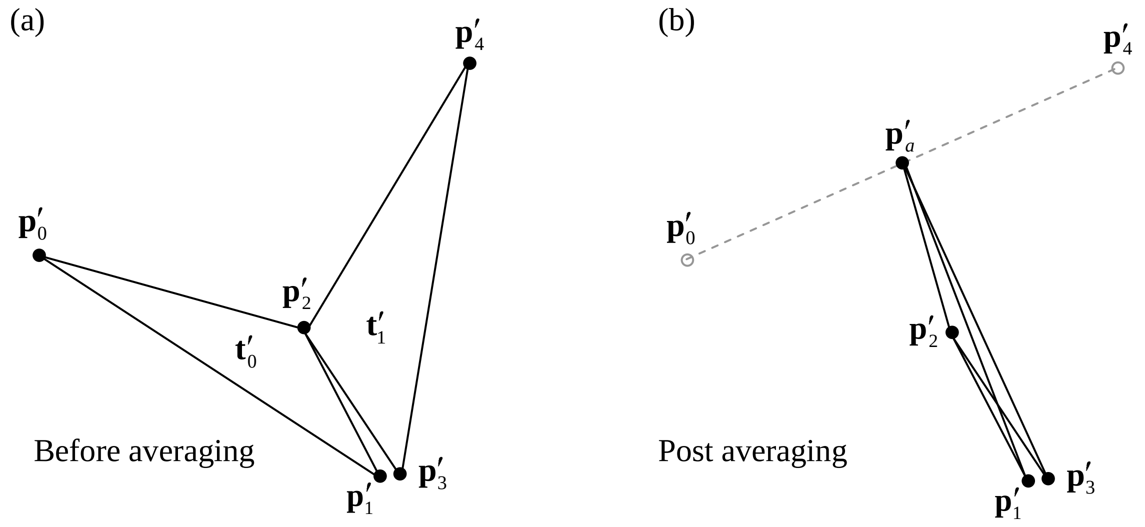


Figure 4.8 Highly curved surfaces problem. (a) Triangles before and (b) after averaging.

This inconsistency in the vertex orientations means that even after the attempt has been made to reconnect the triangles, it will still produce an invalid topology. Furthermore, overlapping of triangles will remain within the triangulation because of this orientation inconsistency, as shown in Figure 4.8(b), which is highly undesirable. This invalid topology will mean that the algorithm will no longer operate correctly and cause a spiralling effect as shown in Figure 4.9. The result will be an unacceptable flattening.

(a) (b)

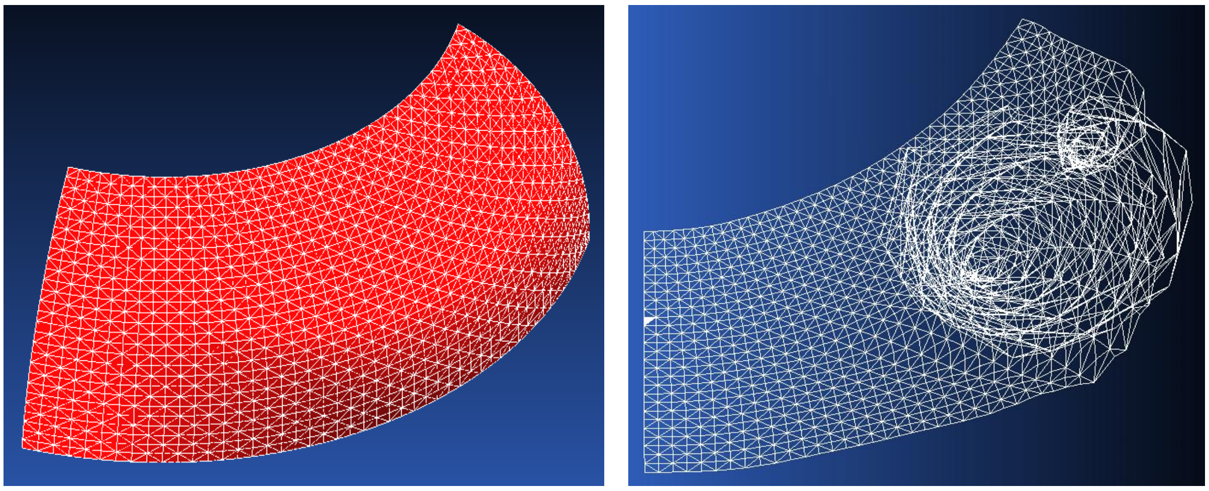


Figure 4.9 (a) Initial 3D triangulation. (b) Failed flattening (solid white triangle is seed).

4.4.2 Long Thin Triangles Problem

As Cader *et al.*'s approach has been optimised for almost developable surfaces, it may suggest that the spiralling effect caused by highly curves surfaces would not occur; however, this is not the case as it can also be induced by triangles with high aspect ratios. Numerical calculations based on the vertices of triangles with high aspect ratios can be unstable. Thus, during the averaging process such triangles can become highly susceptible to change of orientation as shown in Figures 4.10. Figure 4.10 is similar to the case shown in Figure 4.8, however, it is noted that triangles with high aspect ratios are more unstable and can cause vertex orientations to flip with small changes in vertex positions.

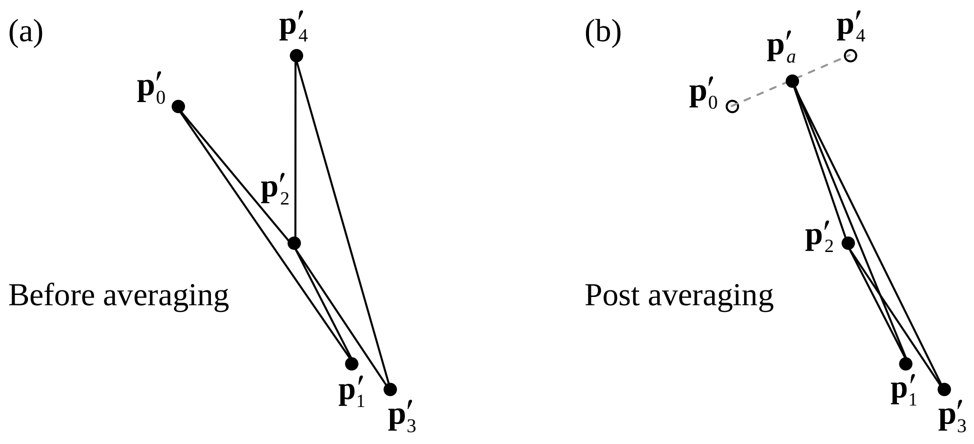


Figure 4.10 High aspect ratio problem. (a) Triangles before and (b) after averaging.

Therefore, a flattening success criterion is established so that, if during a flattening simulation, a triangle flips from anti-clockwise to clockwise, the simulation is immediately terminated and deemed a failure. Furthermore, a definition for 'almost developable' is proposed based on this triangle orientation flipping. Given a non-developable surface, multiple flattening simulations of this surface can be performed by using each triangle within the 3D triangulation as a seed triangle. If, for any seed triangle, the flattening simulation is completed successfully with no triangle orientation flipping, then the surface is deemed as 'almost developable'. Otherwise, if every flattening simulation failed, due to a triangle orientation flip, then the surface is not considered as 'almost developable'.

4.4.3 Flattening Algorithm Stability Solution

The problem of vertex orientation flipping caused by triangles with high aspect ratios suggests that when flattening a trimmed surface, it is best to flatten the entire underlying surface as a rectangular patch with nicely shaped triangles. The trimming curves can then be mapped onto the 2D flattening. This will ensure that triangles with high aspect ratios, which generally occur along the boundary edge of a trimmed surface triangulation, are not dealt with during the flattening process. This can greatly enhance the stability of the flattening algorithm.

4.4.4 Seed Triangle and Mapping Order

McCartney *et al*. and Cader *et al*. noted that the choice of seed triangle can affect the final flattening. As the flattened seed triangle geometry is constrained, any distortion spreads outwards from this seed triangle. This can be seen illustrated in Figure 4.11, where the 3D surface (Figure 4.11(a)) has been flattened twice using two different seed triangles (indicated by the solid white triangles in (Figures 4.11(b)-(c)), resulting in completely different flattening results.

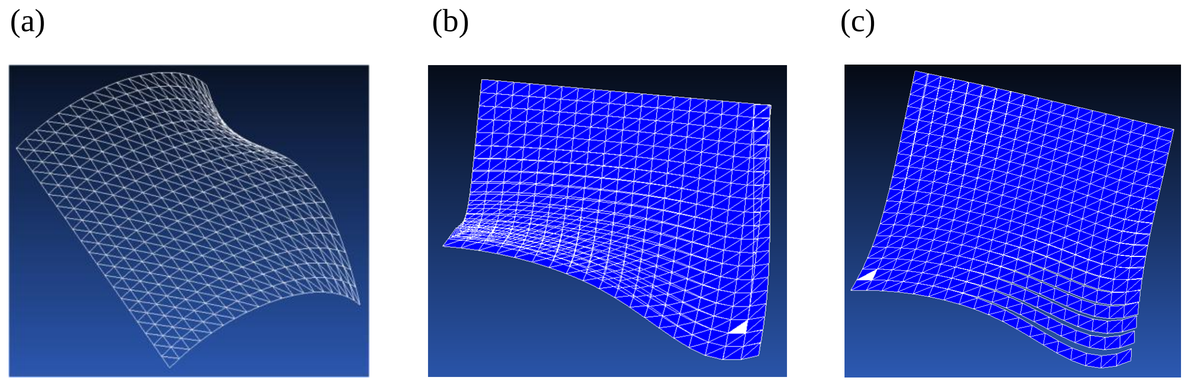


Figure 4.11 (a) 3D triangulation. (b) Result 1. (c) Result 2.

It can be useful, during a flattening simulation, to find the optimum seed triangle that causes the least change in area between the original 3D triangulation and the flattened counterpart. This essentially approximates the change in surface area, which can be a useful indicator for checking the quality of flattening result (Wang *et al*., 2002). Therefore, an optimum seed triangle can be found by performing multiple flattening simulations on a single 3D triangulation, each time using a different triangle as the seed. The areas change induced can be recorded and then compared to find which seed triangle caused the least difference, i.e. making it the optimum seed.

Cader *et al*. also noted that an aspect that can affect the final output is the mapping order and therefore established a consistent method for pre-determining the order in which the triangles will be flattened. This ensures a consistent implementation of this technique by ensuring that given a seed triangle, the flattening result will always be the same.

Chapter 5

RIGHT ANGLE TRIANGLE CONFIGURATIONS

Structured triangulation schemes for approximating trimmed surfaces are widely used throughout CAE as an input for downstream applications, including surface flattening (McCartney *et al*., 1999; Cader *et al*., 2005). The advantage of structured triangulation schemes is that any irregular shaped triangles, particularly those with high aspect ratios, are confined to the boundary edges of the surfaces, whilst maintaining a uniform triangulation within the interior of the surface. This can be beneficial to applications, such as surface flattening, where triangles with high aspect ratios can cause problems during the course of a simulation (Section 4.4).

Apart from along the boundary edges, in general, structured triangulation schemes tend to use right angled triangles (RATs) (Rockwood *et al*., 1988; Peterson, 1994; Kumar and Manocha, 1996; Piegl and Tiller, 1998; Sadoyan *et al*., 2006), as they are easily generated from the cell data structures that are used to partition a surface's parameter domain and establishing which portions of the surface are within the trimming region (Section 2.4). However, RATs offer an interesting variable in terms of their connectivity which can change the final result of a surface flattening simulation. This novel connectivity issue will be the main focus of this chapter.

5.1 Right Angle Triangle Configuration Definition

Given a surface parameter domain which has been sampled to generate a triangulation consisting of equilateral triangles, using all the vertices, there is only one set of equilateral triangles that will fit this point sampling (Figure 5.1(a)).

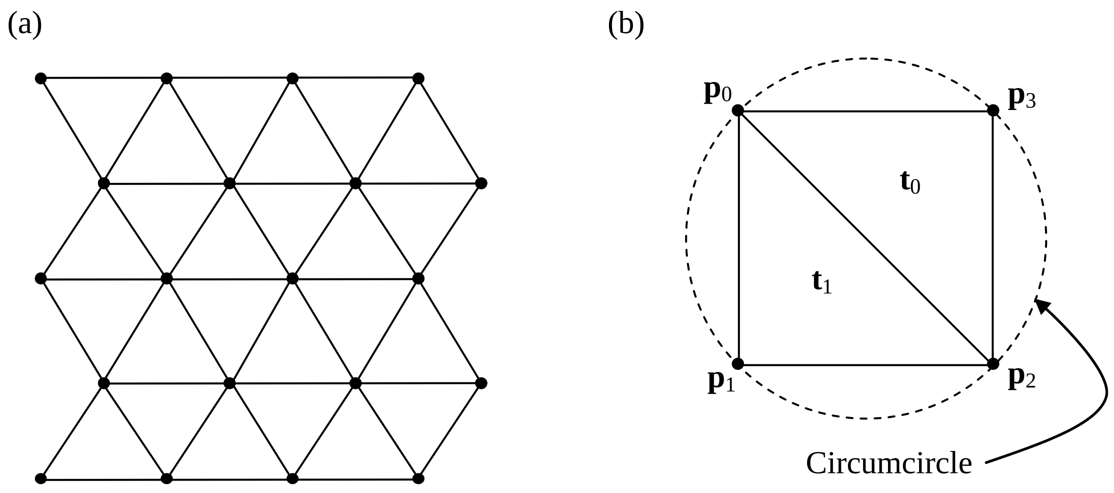


Figure 5.1 (a) Triangulation consisting of equilateral triangles. (b) Circumcircle of each right angle triangle, $\mathbf{t}_0$ and $\mathbf{t}_1$, has four points on it.

The Delaunay triangulation method (Section 2.2) triangulates a set of points by maximising the minimum angle of all the triangles. This method is said to create a unique solution (O’Rourke, 2001) as it is attempting to make the triangles as close to equilateral as possible, which is known to have a unique solution (de Berg *et al.*, 2000). However, there is ambiguity in this method for the situation where more than three points lie on the circumcircle of a triangle (Sugihara and Inagaki, 1995) (Section 2.10). Therefore, a uniformly distributed grid of 2D points would not yield a single solution using Delaunay triangulation as each triangle would have a minimum of four vertices on its circumcircle (Figure 5.1(b)).

Uniformly sampled grids of points are commonly used in many structured triangulation algorithms (Sections 2.4-2.9, Section 3.1). These sampled points are connected to form rectangular cells, which are then triangulated into RATs by adding diagonals. The placement of the diagonal is not unique as there are two choices (Figure 5.2). In both cases the triangles generated by inserting a diagonal will be RATs.

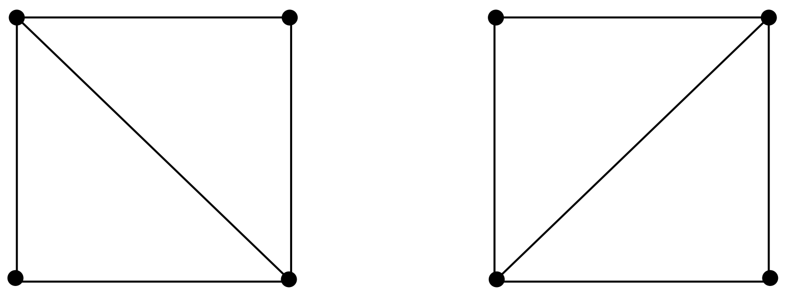


Figure 5.2 Diagonal placement.

As there are only two solutions for the placement of the diagonal from a single cell, the number of permutations for the diagonal placement in *n* cells can be modelled as a binary permutation. Therefore, the number of right angle triangle configurations (RATCs) that can be generated from a grid consisting of *n* cells is:

$$\textit{Number of RATCs} = 2^n \tag{5.1}$$

As *n* increases, the number of RATCs increases exponentially, which can result in an unmanageable number of different triangulations being generated from a uniformly sampled grid, all consisting of RATs. As an example, 100 cells would generate $1.26765e^{30}$ different RATCs. Typical structured triangulations consist of many hundreds, even thousands of cells which must be triangulated in RATs. Suggesting a method that would generate every possible RATC for a uniformly sampled grid of points and then checking the flattening results for each RATC would not be sensible. For example, given a triangulation consisting of 200 triangles (generated from 100 cells), if it takes 1 second to find the optimum seed for each RATC, then the time taken to find the optimum seed for all the RATC permutations of this point sampling (neglecting the time taken to generate the RATC) would be $4e^{22}$ years. In reality, finding the optimum seed triangle and form each RATC could take several minutes, taking into account the density of the triangulation, making this an even more unrealistic task. This motivates the investigation into whether different RATCs affect surface flattening and attempting to establish which RATC would be best suited to this downstream application.

5.2 Global Right Angle Triangle Configurations

In practice, a small number of RATCs are actually used to triangulate a grid of points, and these tend to be the same, within four incident cells, over the whole triangulation, i.e. globally. Two global RATCs were noted from previous work on surface flattening, and a novel third configuration is proposed. These three global RATCs will be used throughout the analysis and testing process in this thesis and are defined as:

1. Regular configuration, used by McCartney *et al*. (1999) and Wang *et al*. (2002) (Figure 5.3(a)).
2. Diamond configuration, used by Cader *et al*. (2005) (Figure 5.3(b)).
3. Chevron configuration (Figure 5.3(c)).

All three configurations are legal RAT triangulations generated from the same point sampling, and each produces a different flattening result as demonstrated in Section 5.2.1.

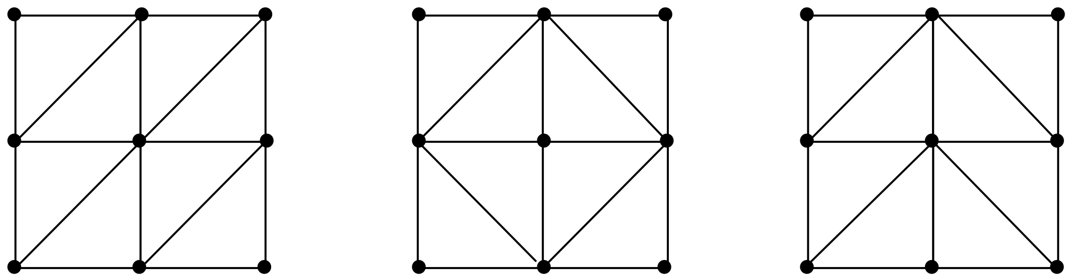


Figure 5.3 Global right angle triangle configurations.

5.2.1 Global RATC Test

To consider the effects of RATC on the flattening process, a non-developable NUBS test surface (Figure 5.4) was sampled uniformly in the parameter domain using a 60×60 grid of points and triangulated in each of the three different global configurations. This point sampling ensured that the area of the model approximated by the triangulation was within 1% of the original surface area value.

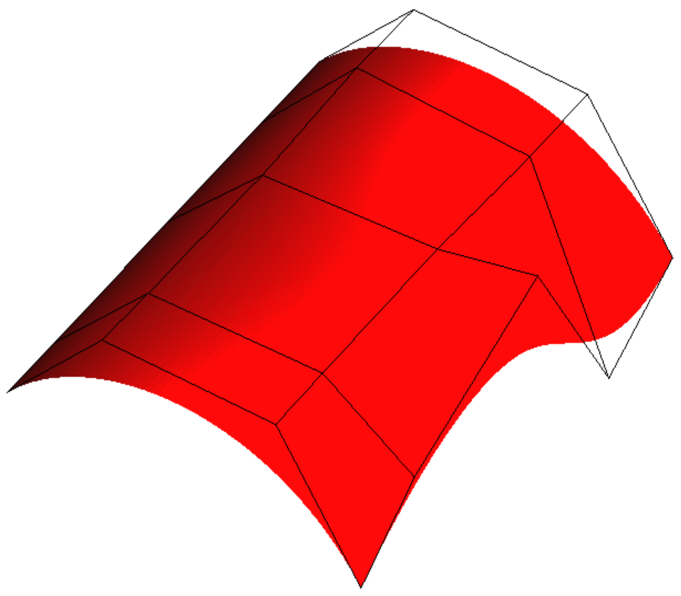


Figure 5.4 Test surface.

Each triangulation consisted of 6962 triangles, which was flattened using Cader *et al.*'s (2005) flattening method and the same seed triangle. The choice of seed triangle can change the result of a flattening simulation (Section 4.4.4); therefore the seed used in the tests was specifically chosen because it was defined by the same vertices and edges in each of the three configurations, thus giving an unbiased starting point to each flattening simulation. The flattening results for the three triangulations are given in Table 5.1 and Figures 5.5-5.7. The seed triangle is shaded in solid black to show its location.

Number of triangles in each triangulation 6962				
Global RATC	No. of triangles flattened	Area Change (δA)	Shape Change (δS)	Successful
Diamond	5903	-15.23%	-15.74%	No
Chevron	6962	-0.27%	-0.05%	Yes
Regular	6962	-0.18%	-0.08%	Yes

Table 5.1 Global RATC test flattening results.

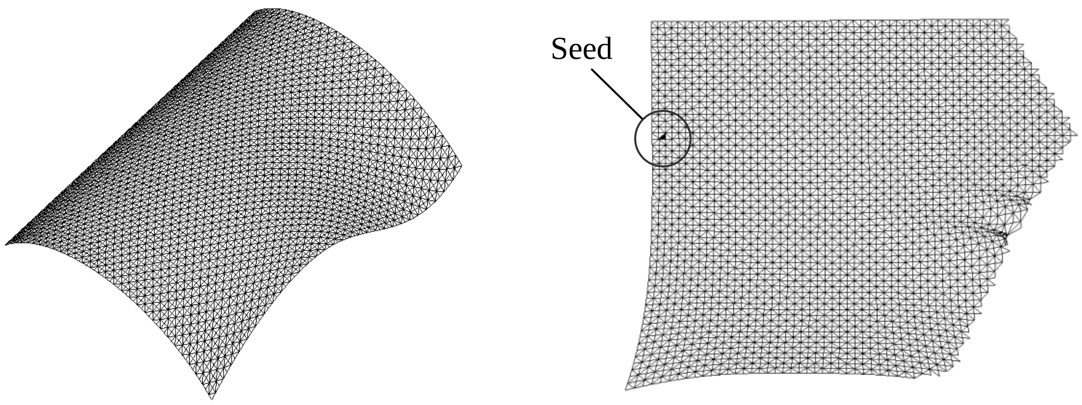


Figure 5.5 (a) Diamond RATC 3D triangulation. (b) Flattening result.

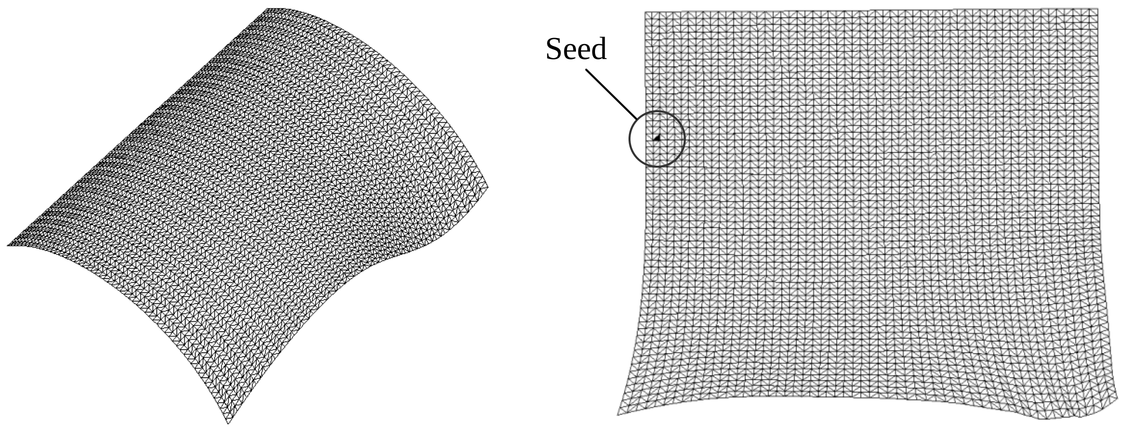


Figure 5.6 (a) Chevron RATC 3D triangulation. (b) Flattening result.

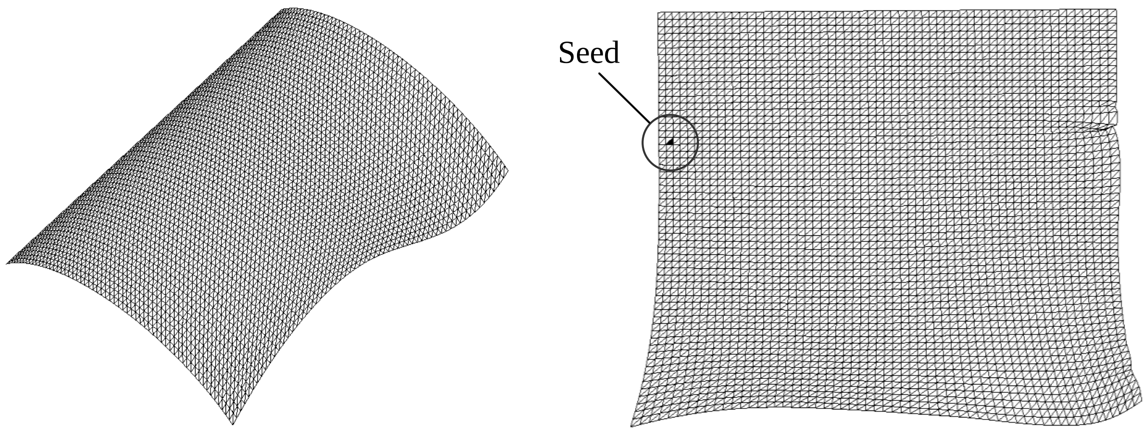


Figure 5.7 (a) Regular RATC 3D triangulation. (b) Flattening result.

As can be seen from Table 5.1, the flattening process is dependent on the global configuration of the triangulation. In the diamond configuration, only 82% of the triangles were processed before the flattening failed. This was caused by a triangle orientation flipping to clockwise during the averaging process (Section 4.4.1). In contrast, the chevron configuration resulted in the triangulation flattening successfully with an area change of -0.27%, where the negative value indicates a reduction in area. An unmodified version of this flattening would cause the triangulation to have overlapping triangles or gaps between triangles. However, the averaging process removes overlaps and tears, thus triangle areas can increase or decrease in these regions. The flattening was further improved by using the regular configuration, which reduced the area change by 33% compared to the chevron configuration.

It can be seen from Figure 5.7, that whilst the regular configuration induced less change in area, it produced some noticeable creasing in the top right corner of the final flattening. In contrast, the chevron triangulation resulted in visually less creasing (Figure 5.6). This is reflected in the resulting shape change coefficients, which were -0.08 and -0.05 for the regular and chevron coefficients respectively, where the negative value indicates a compression of the original edges. This implies that for this particular surface and seed triangle, the chevron configuration preserves the original shape better than the regular configuration.

5.3 Motivation for a Consistent RATC

Currently, the failure of a surface flattening of this nature would be solely attributed either to the type of triangulation, i.e. triangle shape, triangle size, or flattening algorithm. However, it has been shown that the performance of the flattening algorithm is also dependent on the configuration of RATs.

When using a triangulation for a downstream engineering application, much effort is put into the creation of the application algorithm to ensure that any configuration ambiguities are removed. For example, Cader *et al*. (2005) noted that a flattening simulation could be affected by the mapping order of the 3D triangles, therefore an edge based mapping order was introduced (Section 4.2.2) to remove this variable. However, despite the measures put into place to make flattening algorithms more robust, they are still triangulation dependent algorithms. Therefore, to achieve a more robust flattening simulation, efforts must be made to remove any ambiguities from the initial 3D triangulation, i.e. RATC.

Section 5.2.1 showed that different RATCs can have an impact on the resulting surface flattening. However, the question of which RATC is best suited for surfaces that require flattening still remains. It would appear that the chevron and regular configurations provide a better mapping than that of the diamond configuration, however, which of these two configurations is better suited for flattening is unclear. Furthermore, it will be worth investigating whether a RATC that varies locally, depending on the local characteristics of the surface, would be better suited than a global configuration.

In the field of discrete data triangulation and remeshing (Frey and George, 2000; Meek and Walton, 2000; Dyn *et al.*, 2001), triangulation choices, made from a given set of points, are determined by locally evaluating the curvature (generally Gaussian or mean) from the mesh to ensure that the distribution of curvature is smooth over the entire mesh. Although making the Gaussian curvature smooth over the triangulation does not necessarily guarantee a good approximation, the idea of matching a triangulation to better approximate a surface characteristic is a topic which is worth exploring.

Therefore, the remainder of this chapter will firstly present a method for generating a locally varying RATC, within a triangulation, to best match the Gaussian curvature of the original surface. This local RATC and the three global RATCs (Section 5.2) will be applied to the triangulations of two almost developable surfaces. The results will then be analysed and compared to establish which type of RATC is best suited for surface flattening.

5.3.1 Surface Approximation

In Section 5.2.1, each RATC was generated from the same set of points, yet still produced varying flattening results. One of the contributing factors that could have affected the results could be the quality of the original surface approximation by the triangulation.

Each RATC inherits positional accuracy from the surface because the triangle vertices are guaranteed to lie on the surface; however, the connectivity of these points (i.e. placement of triangle edges between points) could affect the approximation of other geometric characteristics. For example, let **p** be a vertex in a 3D triangulation, which is incident to the triangles $\mathbf{t}_i$, where $i = 0, 1, \ldots, n$. To estimate the unit normal at **p**, the unit normal of each $\mathbf{t}_i$ (Section 1.6.7) would need to be generated first (Figure 5.8) and then averaged to estimate the unit normal vector at **p**. As triangles are planar polygons, different configurations of triangles incident to **p**, could yield different values for the unit normal vectors for each triangle. This is illustrated in Figure 5.8 where changing the configuration from regular to the chevron altered the triangle unit normal vectors shown in blue. Once averaged, these unit normal vectors may give different values at **p**.

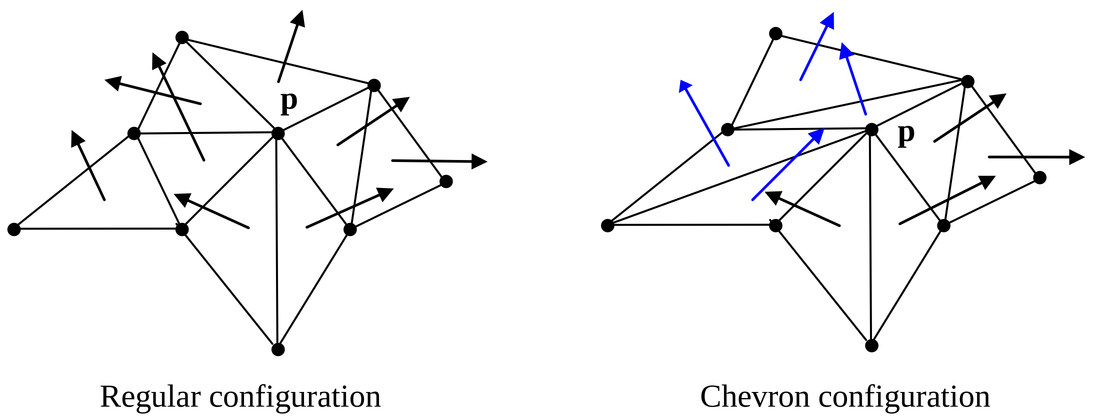


Figure 5.8 Triangles incident to **p** and normal estimations.

When generating points, directly from a surface, little effort is put into ensuring that the triangulation configuration approximates the original surface characteristics as well as possible. This motivates the investigation into finding a RATC that will best approximate the geometrical characteristics of the original surface.

Therefore a greedy algorithm (Cormen *et al.*, 2001) is proposed to establish the optimum RATC that will best approximate a specific surface characteristic, known as the RATC controller. The algorithm will find the optimum RATC, for a triangulation generated from a uniformly sampled grid of points, which best approximates the RATC controller of the original surface. By doing this for each RATC controller, a well defined RATC solution will be generated from any triangulation generated from this data set, which will vary locally, and remove any ambiguities which could be induced into any flattening simulations. These triangulations will also provide a premise to evaluate whether a global configuration or locally distributed RATC configuration provides a better flattening result.

The surface characteristic that will be used to the control the RATC configuration of a triangulation will be the Gaussian curvature, as this measure is commonly used to evaluate the shape of 3D surfaces (i.e. developable surfaces) in surface flattening (McCartney *et al.*, 1999; Cader *et al.*, 2005). Details for approximating this characteristic from a triangulation will be provided in Sections 5.5.

5.4 Optimised RATC Algorithm

The optimised RATC is defined as the configuration that best approximates the Gaussian curvature of the original surface at the sampled points. The greedy algorithm created for determining the optimum RATC is detailed below.

Stage 1: Subdividing the Surface

Given a NUBS surface, $\mathbf{S}(u, v)$, defined over the normalised parameter domain $u,v \in [0,1]$ (Section 1.2.1), the first stage is to represent the parameter plane as a uniformly subdivided $m \times n$ grid of points in the u and v directions respectively. The uv grid points and their indices are organised as shown in Figure 5.9.

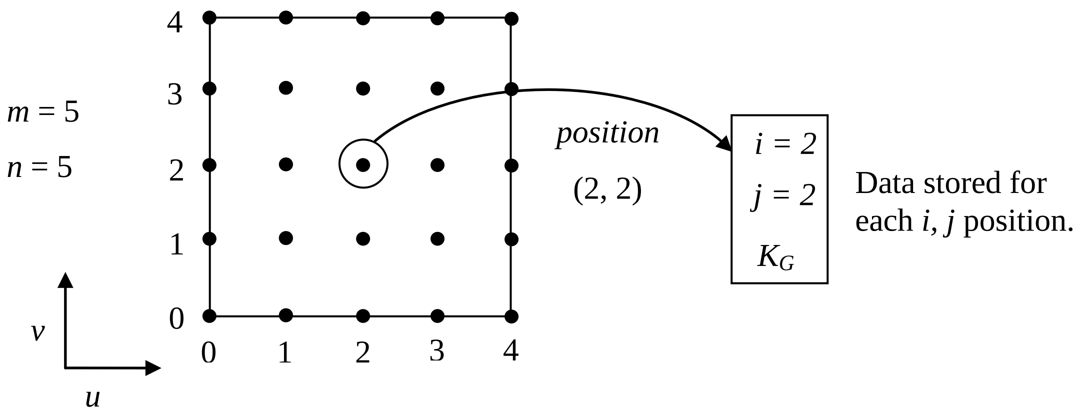


Figure 5.9 Nomenclature and data stored for each *uv* point in a grid.

Stage 2: Calculate Surface Gaussian Curvature Values

The $\mathbf{S}(u, v)$ is then evaluated at each *uv* grid point to find the Gaussian curvature, K_G, at that parameter value. Each K_G is stored for the appropriate *uv* grid point as shown in Figure 5.9. During this stage, an additional check will be performed to establish the *uv* grid point that has the largest magnitude of Gaussian curvature. This *uv* grid point will be stored as $\mathbf{p}_g(i_{max}, j_{max})$, and will form the initial starting point for the greedy algorithm. This is done to ensure that the RATs are configured to estimate highly curved regions of the surface more accurately, as these have more impact on the flattening simulation.

Stage 3: Generating Optimum RATC

The method starts from *uv* grid point $\mathbf{p}_g(i_{max}, j_{max})$. The 8 adjacent vertices are extracted from the subdivided grid (Figure 5.10).

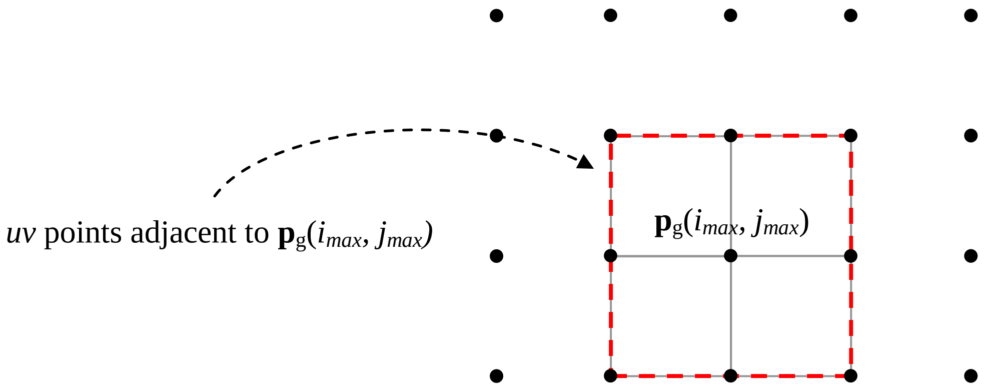

uv points adjacent to $\mathbf{p}_g(i_{max}, j_{max})$
$\mathbf{p}_g(i_{max}, j_{max})$

Figure 5.10 Grid points adjacent to $\mathbf{p}_g(i, j)$.

These vertices are mapping into 3D and stored as four cells, which are then triangulated 16 times, each time using different combinations of diagonal placements between the cells (Section 5.1). For each triangulation, the K_G approximation will be calculated at the 3D mapped point of $\mathbf{p}_g(i_{max}, j_{max})$ (Section 5.5). The triangulation configuration that produces the least absolute difference between the actual and approximated K_G will be chosen as the local configuration for these points.

The remainder of the points are then processed in rows, which is outlined in Stage 4, and then this process is repeated for the next point. However, it is worth noting that as this is a greedy algorithm, only the first *uv* grid point will have to test 16 different configurations. For example, when the second point is being processed, two of the four cells will have already been triangulated (Figure 5.11). Therefore, only two cells will have the freedom to be tested for different configurations, meaning that there will only be 4 (i.e. 2^2) new configurations that will need to be checked.

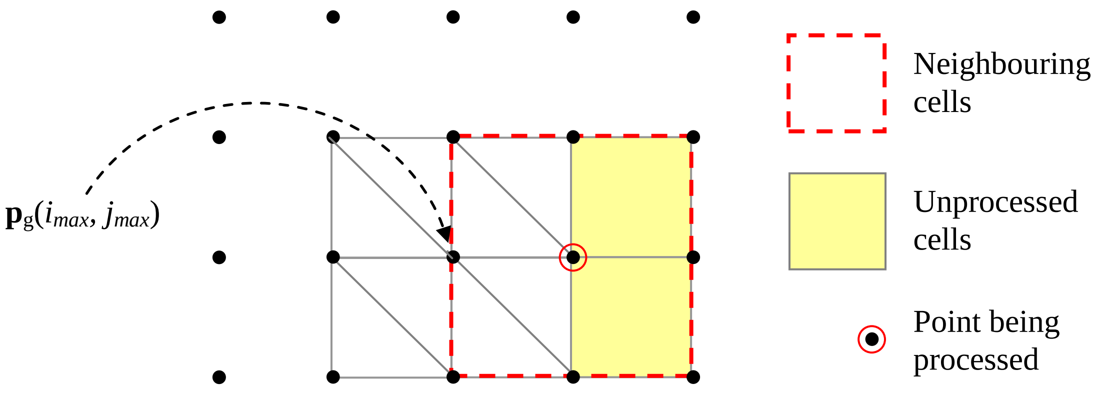


Figure 5.11 Grid points adjacent to $\mathbf{p}_g(i, j)$.

Stage 4: Points Processing Order

Once the optimum local configuration has been found for $\mathbf{p}_g(i_{max}, j_{max})$, the remaining *uv* points are processed, using the method shown in Stage 3, incrementing *i* by 1, while $i < m$. Once a row has been processed the next row to be processed will be j+1, while $j < n$. However, as this method starts from index (i_{max}, j_{max}), increments of i+1 and j+1 will only process grid points with larger index positions than (i_{max}, j_{max}). Therefore, a second pass, which travels in the opposite direction, must be carried out to process *uv* grid points with lower indices (Figure 5.12). The increments will be i-1, while $i > 0$ and then j-1, while $j > 0$. Stages 3 and 4 are repeated until the entire grid of points has been processed.

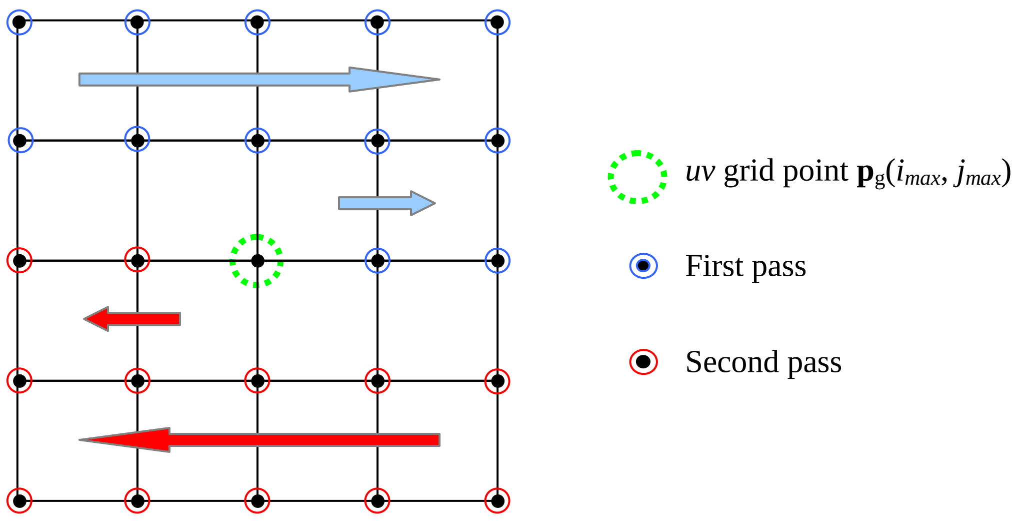


Figure 5.12 Grid point processing directions.

5.5 Gaussian Curvature Approximation

The Gaussian curvature is a measure commonly used in surface flattening (McCartney *et al.*, 1999; Cader *et al.*, 2005) and remeshing (Meek and Walton, 2000; Dyn *et al.*, 2001). This section will present a method which is commonly used to calculate an approximation of Gaussian curvature from a surface triangulation.

5.5.1 Current Gaussian Curvature Approximation

Given a triangulation, **T**, the Gaussian curvature at the point, **p**, can be estimated by using the triangles incident to **p** (Dyn *et al.*, 2001).

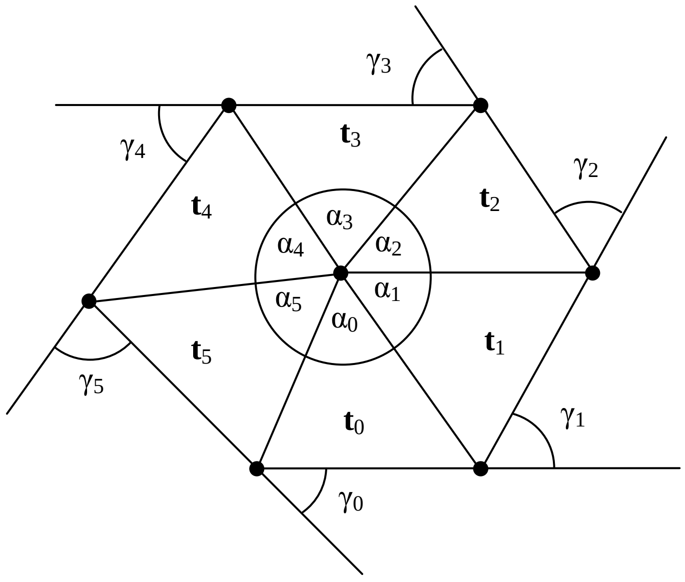


Figure 5.13 Nomenclature for Gauss-Bonnet estimation.

Let $\mathbf{t}_i$, $i = 0, 1, \ldots, n$, be the triangles incident to **p**, in anti-clockwise order, and let α_i be the interior angle of triangle $\mathbf{t}_i$, incident to **p** (Figure 5.13). Also, let γ_i be the outer angle made between successive triangles ($\mathbf{t}_i$, $\mathbf{t}_{i+1(\mathrm{mod}\ n)}$) edges that are opposite to **p**. This gives

$$\sum_{i=0}^{n} \alpha_i = \sum_{i=0}^{n} \gamma_i \tag{5.2}$$

The Gauss-Bonnet, in the topological case, reduces to

$$\iint_A K_G dA = 2\pi - \sum_{i=0}^{n} \gamma_i = 2\pi - \sum_{i=0}^{n} \alpha_i \tag{5.3}$$

where the total area of the triangles is

$$A_{\mathbf{T}} = \sum_{i=0}^{n} \Delta_i \tag{5.4}$$

and Δ_i is the area of triangle $\mathbf{t}_i$. Therefore, assuming the curvature is constant in the local neighbourhood, the Gaussian curvature, K_G, is given by

$$K_G = \frac{2\pi - \sum_{i=0}^{n} \alpha_i}{A_{\mathbf{T}} / 3} \tag{5.5}$$

where $A_T/3$ is equivalent to the barycentric area of the triangles (Section 1.6.6).

5.5.2 Approximation Method Limitation

This Gaussian curvature approximation algorithm is commonly used in remeshing and discrete data triangulation (Frey and George, 2000; Meek and Walton, 2000; Dyn *et al.*, 2001). In general, these methods are operating on smooth data and triangulations that define equilateral triangles, which ensure that every vertex within the triangulation has a valency of six. However, a consistent vertex valency, throughout an entire triangulation, is not necessarily guaranteed when using RATs. Therefore, (5.5) will not necessarily work very well for triangulations consisting of RATC with varying valencies, due to the area portion of the algorithm.

To calculate the Gaussian curvature local to a point **p,** using (5.5), the barycentric area (Section 1.6.6) is required. This area ensures that only the local area to the vertex **p** is taken, thus minimising the amount of bias from other neighbouring vertices. However, this local area value can change when using RATs, which can change the Gaussian curvature estimation at **p**. Consider 9 planar vertices, where each cell has unit area as shown in Figure 5.14.

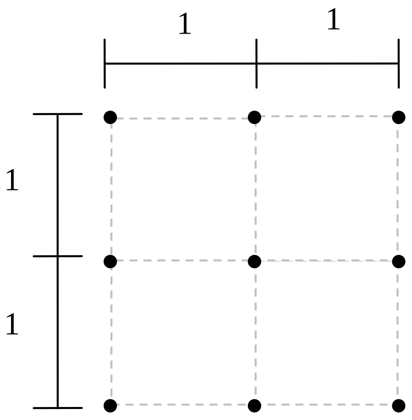


Figure 5.14 Nine vertices where the area for each cell is 1.

For these points, there is a possibility of 16 different RATC that could be generated, however, only the 3 shown in Figure 5.15 will be considered, as these show the minimum (4), median (6) and maximum (8) valencies which can be generated from the point sampling at **p**.

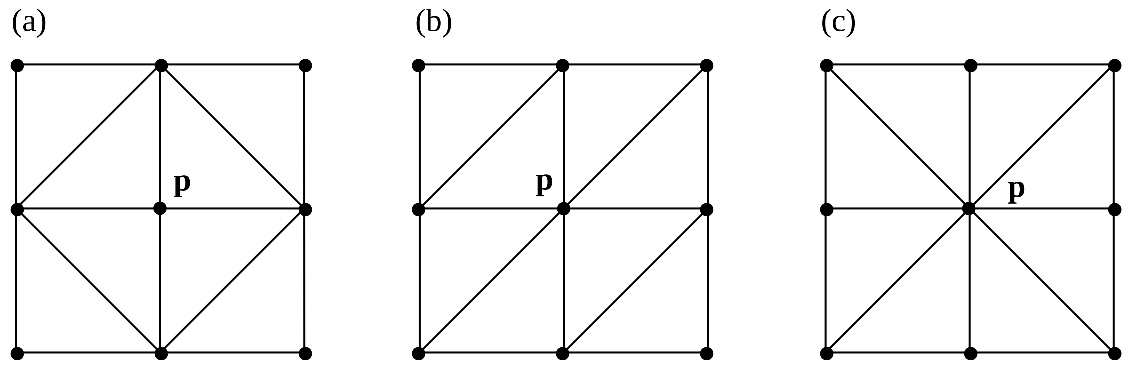


Figure 5.15 RAT triangulation with valency (a) 4, (b) 6 and (c) 8.

In each of these 3 configurations, the sum of the internal angles would be the same; however, they would generate different values for the local triangulation and barycentric areas as shown in Table 5.2.

	Triangulations from Figure 5.15		
	(a)	(b)	(c)
Valency of **p**	4	6	8
Total Local Area	2	3	4
Barycentric Area	0.67	1	1.33

Table 5.2 Triangulation Areas.

For almost planar vertices, the sum of the internal angles for each configuration would be similar; however, the changes in the barycentric area would cause differences in the Gaussian curvature approximation using (5.5) at the vertex **p**. The discussion of which RATC valency would produces the best approximation using (5.5) is given below.

5.5.3 Area Errors

Given a set of points, sampled for RATs, the optimum local area which should be approximated to ensure that the area is distributed evenly across all points, would be the square areas shown in Figure 5.16, where the square around **p** would be generated by using the centre points of the four adjacent cells to **p**.

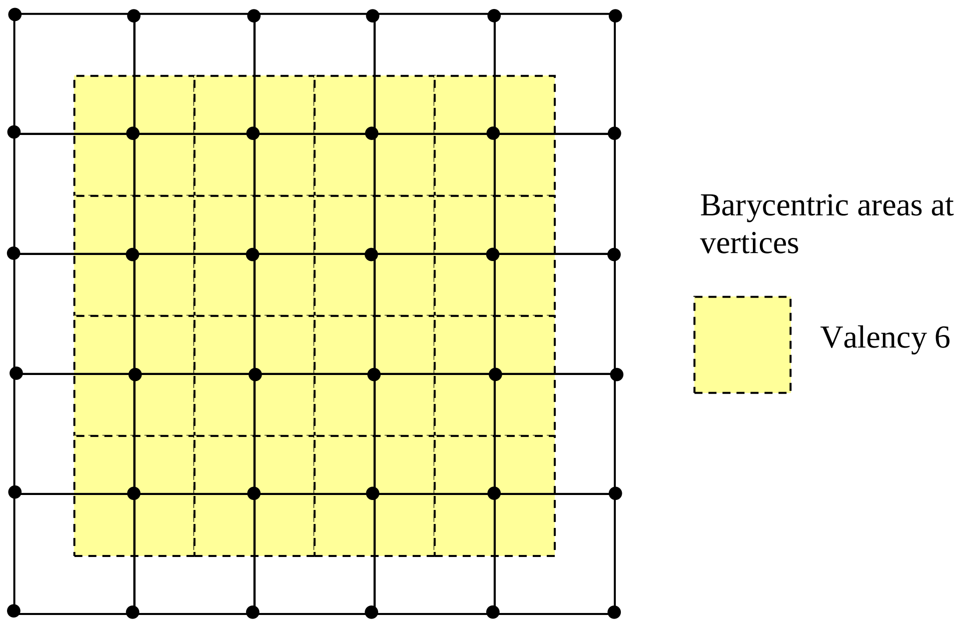


Figure 5.16 Unbiased local area partitioning of a uniformly sampled grid.

Assuming that all the points have the dimensions shown in Figure 5.14, the local area around each point **p**, which would give an unbiased area distribution, would be 1. It can be seen from Table 5.2, that the only configuration which would meet this unit area requirement for the barycentric area, is where the valency of the points was 6. A valency of 4 gives less area (i.e. larger Gaussian approximation) and a valency of 8 gives a greater area (i.e. smaller Gaussian approximation) (Figure 5.17).

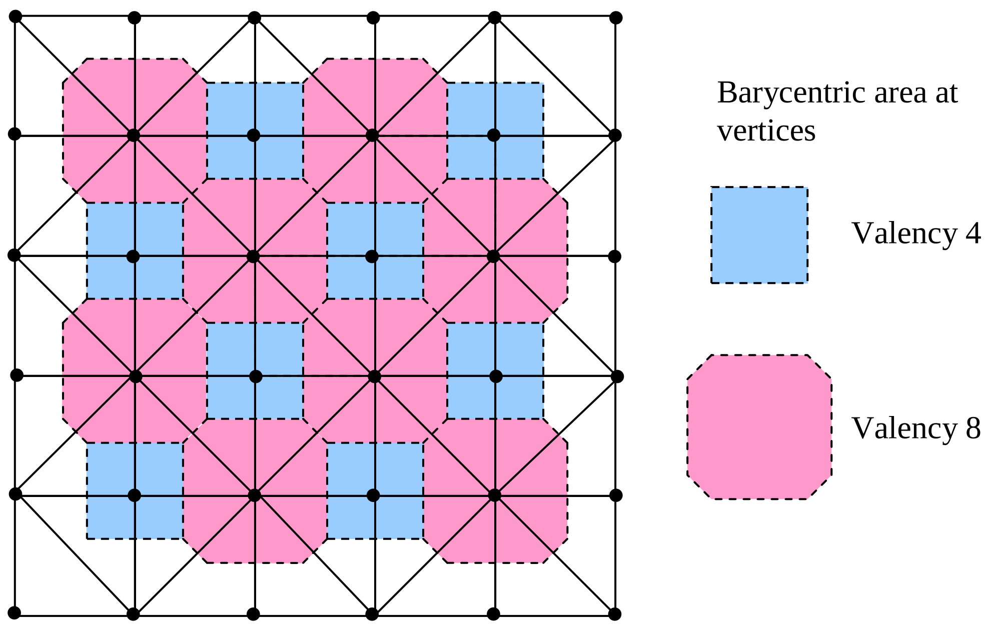


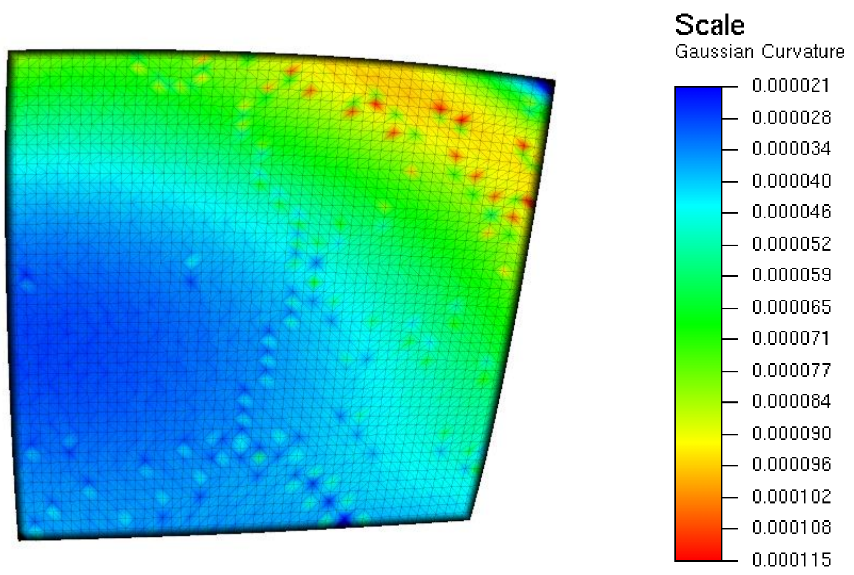
Figure 5.17 Uneven area distributions from a uniformly sampled grid.

5.5.4 Experimental Validation

To validate this theory, the Gaussian curvature was calculated using (5.5) on the triangulation of an almost developable surface. The RATCs were locally varied at random to produce differing valencies. The surface was taken from a trimmed surface definition of an industrial model defining a section of a car backlight, provided by Pilkington Group Limited (Pilkington, 2010).

Figure 5.18(a) shows the Gaussian curvature approximated from the triangulation and Figure 5.18(b) shows the absolute difference between Gaussian curvature of the actual surface and the equivalent approximation from the triangulation, at the same 3D vertex positions.

(a)



(b)

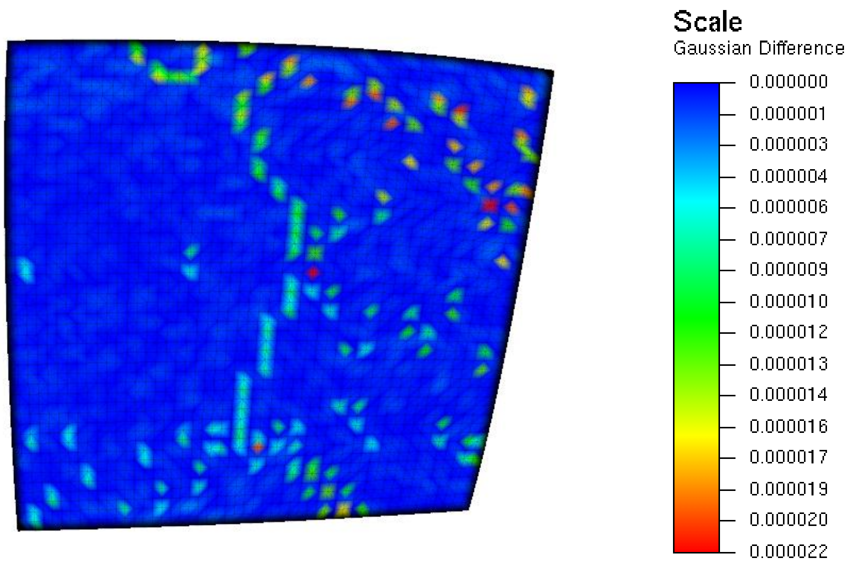


Figure 5.18 (a) Gaussian curvature approximated by the triangulation. (b) Difference between surface Gaussian curvature and approximation from the triangulation.

The minimum and maximum Gaussian curvatures from the actual surface, at the sampled points, were 0.000024 and 0.000097 respectively. However, due to (5.5), the minimum Gaussian curvature approximation was underestimated by 13% and the maximum was overestimated by 19%. Given the relatively low curvature values of the surface, these approximation errors are relatively significant. Also, rapid fluctuations in the Gaussian curvature approximation (Figure 5.18(b)) were seen to occur around points where the valency was not six. This observation was validated by mirroring the RATC of Figure 5.18 with respects to the *v* axis in the parameter domain, then mapping these points into 3D (Figure 5.19), thus changing the location of the valencies.

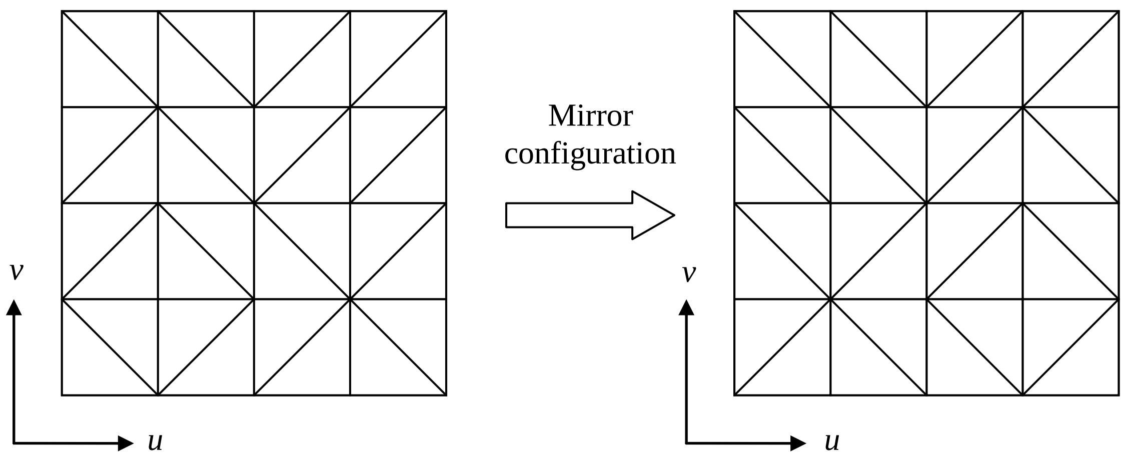
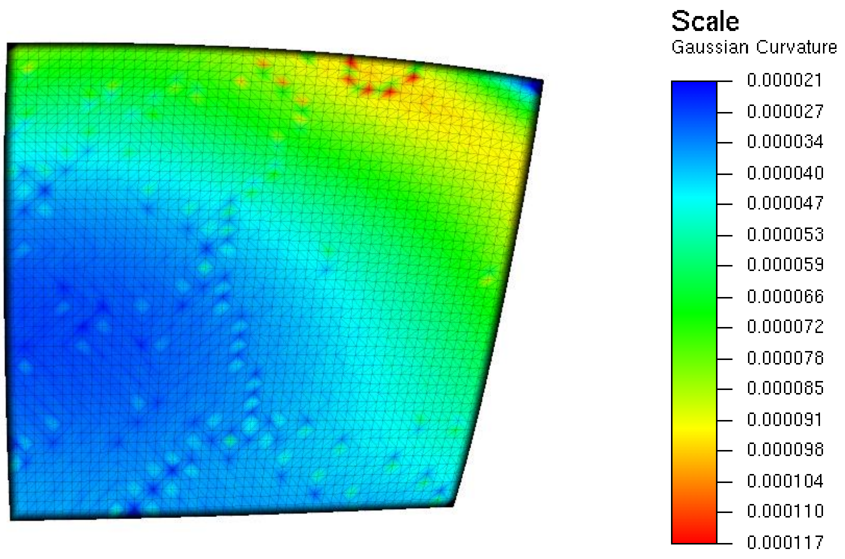


Figure 5.19 Mirroring the RATC in the parameter domain of the surface.

The results of this test illustrated the same results as the first test, in that the Gaussian curvature is approximated best at the triangle vertices that have a valency of six (Figure 5.20).

(a)



(b)

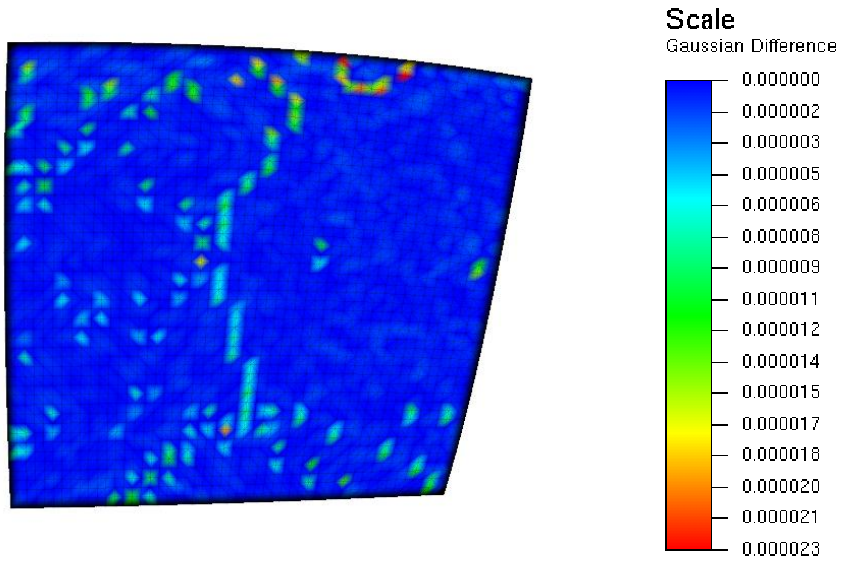


Figure 5.20 (a) Gaussian curvature approximated by the triangulation. (b) Difference between surface Gaussian curvature and approximation from the triangulation.

5.5.5 Improved Gaussian Curvature Approximation

To remove any biases caused by varying valencies within a triangulation, an area compensation factor (A_{CF}) is applied to the area portion of (5.5), which is defined as:

$$A_{CF} = \frac{6}{valency\ of\ \mathbf{p}} \tag{5.6}$$

Therefore, the improved algorithm to calculate the Gaussian curvature, K_G, approximated from a triangulation at vertex **p** is given by

$$K_G = \frac{2\pi - \sum_{i=0}^{n} \alpha_i}{A_{CF} \ A_{\mathbf{T}} / 3} \tag{5.7}$$

where α_i and $A_{\mathbf{T}}$ are defined as in Section 5.5.1.

Applying this area compensation factor has shown to improve the Gaussian curvature approximation generated from a triangulation, when compared against (5.5). To illustrate this improvement, the 3D triangulation shown previously in Figure 5.20 of Section 5.5.4, was re-used to calculate the Gaussian curvature approximation using (5.7). Figure 5.21(a) shows the Gaussian curvature approximated from the triangulation and Figure 5.21(b) shows the absolute difference between Gaussian curvature of the actual surface and the equivalent approximation from the triangulation, at the same 3D vertex positions. The error scale in Figure 5.21(b) is the same as in Figure 5.20(b) to illustrate the relative improvement.

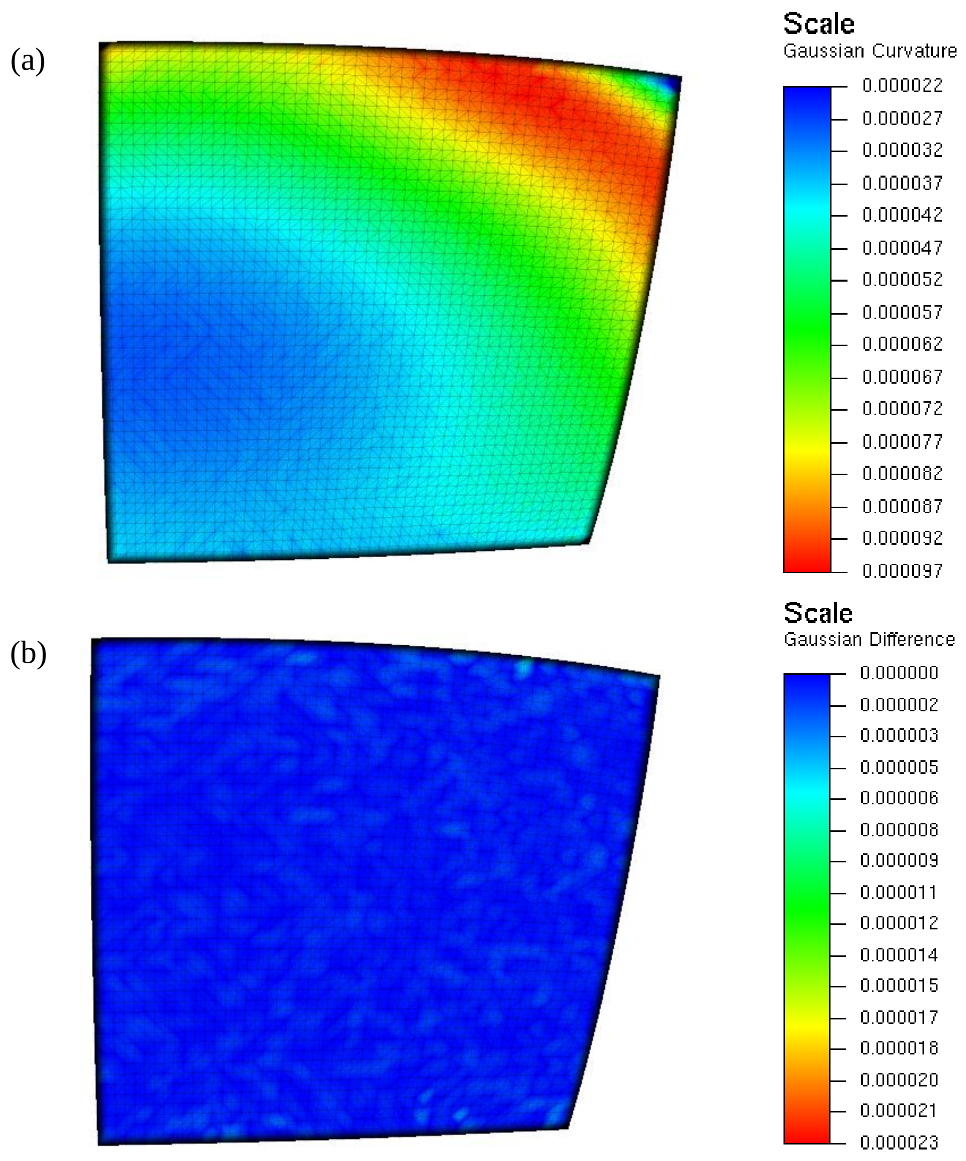


Figure 5.21 (a) Gaussian curvature approximated by the triangulation. (b) Difference between surface Gaussian curvature and approximation from the triangulation.

It can be seen that the occurrence of fluctuations in the Gaussian curvature, caused by the varying valencies, has been reduced. In addition, the accuracy of the minimum and maximum Gaussian curvature approximations has been improved. The minimum curvature was underestimated by 8% and the maximum was calculated exactly, within a tolerance of 1e-6, meaning they corresponded closer to the curvatures of the original surface when using (5.7). In the previous test (Section 5.5.4), the maximum absolute difference between the curvature approximation and the actual curvature value was 0.000023, however, this has been reduced by 79% to 0.000005. This improvement can be seen graphically in Figure 5.21, which has less curvature fluctuations than Figures 5.18(b) and 5.20(b).

5.6 Gaussian Curvature Optimised RATC Testing

This section will test the quality of the Gaussian curvature approximated from a triangulation generated by using the optimised RATC algorithm shown in Section 5.4. Two almost developable surfaces were used during the testing process. Both surface were taken from trimmed surfaces, where one defined a rear light (the same as in Section 5.5), and the other defined a car windscreen. Both surfaces were provided by Pilkington Group Limited. Using the optimised RATC algorithm (Section 5.4), each surface was triangulated to get a RATC that best approximated the Gaussian curvature of the original surface. Using the same point sampling, the surfaces were also triangulated using the 3 global configurations introduced in Section 5.2. The improved Gaussian curvature approximations (5.7) of all triangulation will be compared.

Each triangulation was also put through a flattening simulation to find the optimum seed triangle (Section 4.4.4) that induced the least distortion in terms of the change of absolute area, ΔA (4.9). The optimum seed flattening results will be presented and analysed to establish more insight into the effects of triangulation on flattening.

5.6.1 Results

The actual 3D surface areas for the backlight and windscreen surfaces are 362035mm^2 and 475724mm^2 respectively. To ensure that the surface area approximated by the 3D triangulation was within a 2mm^2 tolerance of the original area, the backlight and windscreen surfaces were subdivided into 60×60 and 70×70 points respectively.

The topological data generated for each RATC was the same for each surface as they used the same initial point sampling and triangle type (i.e. RATs). The triangulation data for each surface definition can be seen in Table 5.3.

Surface	No. Vertices	No.Edges	No. Triangles
Backlight	3600	10561	6962
Windscreen	4960	14421	9522

Table 5.3 Topological data.

The actual surface minimum, K_{Gmin}, and maximum, K_{Gmax}, Gaussian curvatures evaluated at the sampled points are shown in Table 5.4.

Surface	K_{Gmin}	K_{Gmax}
Backlight	0.000019	0.000096
Windscreen	0.000002	0.000089

Table 5.4 Surface Gaussian Curvature.

Tables 5.5 and 5.6 show the Gaussian curvature approximations generated from each RATC triangulation for the backlight and widescreen surface respectively. Each table shows:

1. The minimum and maximum K_G approximations.
2. The minimum and maximum absolute difference between K_G of the actual surface and the equivalent approximation from the triangulation, at the same 3D vertex positions.

3. K_G difference per vertex, which is the average of the total sum of the absolute difference over all the vertices of the triangulation.

Configuration	K_{Gmin}	K_{Gmax}	Min K_G diff	Max K_G diff	K_G diff per vertex
Chevron	0.000013	0.000099	0	0.000006	0.0000008
Diamond	0.000005	0.000102	0	0.000034	0.00000121
Regular	0.000018	0.000098	0	0.000004	0.00000077
RATC Optimised	0.000019	0.000098	0	0.000004	0.00000077

Table 5.5 Backlight triangulation Gaussian curvature approximations.

Configuration	K_{Gmin}	K_{Gmax}	Min K_G diff	Max K_G diff	K_G diff per vertex
Chevron	0.000002	0.000094	0	0.000008	0.00000035
Diamond	0.000002	0.000096	0	0.000009	0.00000061
Regular	0.000002	0.000094	0	0.000008	0.00000034
RATC Optimised	0.000002	0.000092	0	0.000007	0.00000031

Table 5.6 Windscreen triangulation Gaussian curvature approximations.

The optimum seed triangle flattening results for each RATC can be seen in Tables 5.7 and 5.8 for the backlight and windscreen triangulations respectively. For these specific industrial models, an area difference tolerance of 0.1mm^2 was used, as area changes above this tolerance, after flattening, will be considered as significant. Each table shows:

1. Absolute percentage area change between the original 3D and flattened triangulations.
2. The average area change per triangle. This is calculated by summing all area changes greater than a tolerance of 0.1mm^2 and dividing by the number of such triangles.
3. Absolute percentage shape change between the original 3D and flattened triangulations.

4. The average shape change per edge. This is calculated by summing all shape changes greater than a tolerance of 0.1mm and dividing by the number of such edges.
5. The percentage of triangles used as the seed within each triangulation that generated a successful flattening result.

Configuration	ΔA (%)	Average ΔA (mm^2)	ΔS (%)	Average ΔS (mm)	Seed Success (%)
Chevron	0.49	0.405	0.134	0.515	54
Diamond	0.492	0.457	0.135	0.601	34
Regular	0.475	0.4	0.137	0.591	92
RATC Optimised	0.478	0.428	0.128	0.589	30

Table 5.7 Backlight surface triangulation flattening results.

Configuration	ΔA (%)	Average ΔA (mm^2)	ΔS (%)	Average ΔS (mm)	Seed Success (%)
Chevron	0.192	0.220	0.052	0.314	69
Diamond	0.205	0.294	0.057	0.38	26
Regular	0.178	0.209	0.058	0.322	90
RATC Optimised	0.181	0.317	0.055	0.461	20

Table 5.8 Windscreen surface triangulation flattening results.

The flattened triangulations for the backlight and windscreen surfaces, for each RATC, can be seen in Figures 5.22 and 5.23 respectively. Each triangulation also has a colour shading, showing the triangles that had a change in absolute area. The optimum seed triangle positions are shown as solid black and white in the wireframe and colour shadings respectively.

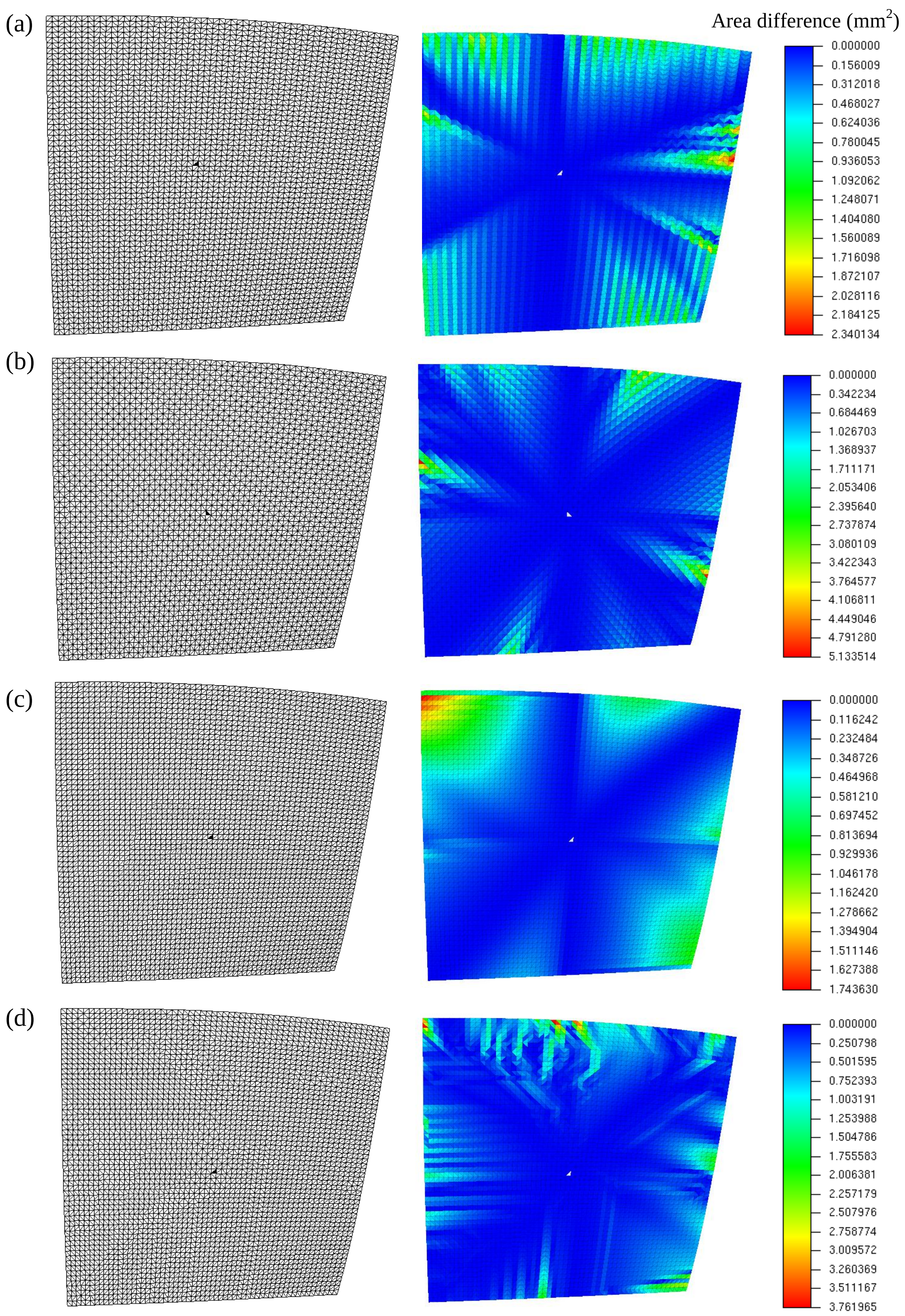


Figure 5.22 Backlight surface flattening results for (a) chevron, (b) diamond, (c) regular and (d) Gaussian curvature optimised RATCs.

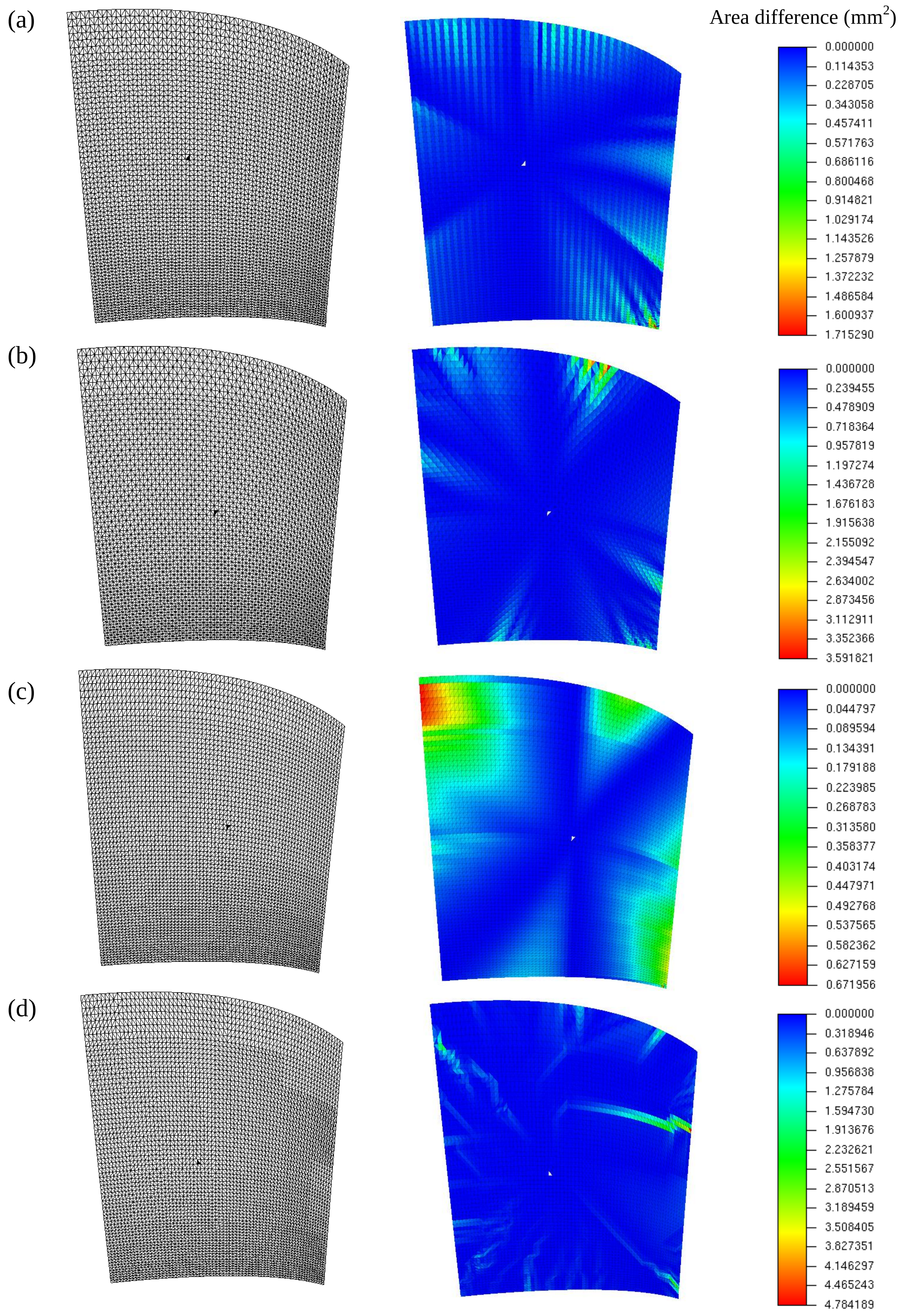


Figure 5.23 Windscreen surface flattening results for (a) chevron, (b) diamond, (c) regular and (d) Gaussian curvature optimised RATCs.

5.6.2 Gaussian Curvature Approximation Analysis

It can be seen from Tables 5.5 and 5.6 that the triangulation which was optimised for the Gaussian curvature generated the closest curvature approximation to the original surface. In general, the curvature approximations generated from all the RATCs were reasonable; however, the diamond RATC generated the most noticeable difference between the configurations. The diamond RATC produced the worst estimates for K_{Gmin} and K_{Gmax}. For the backlight surface, it underestimated K_{Gmin} by 73% and overestimated K_{Gmax} by 6%, and for the windscreen surface, it overestimated K_{Gmax} by 8%. The Gaussian curvature RATC optimised triangulation yielded better results, approximating K_{Gmin} exactly, and overestimated K_{Gmax} by only 2% and 3 % for the backlight and windscreen surfaces respectively.

In addition, the diamond configuration also had the highest average K_G difference per vertex over the entire surface. Figure 5.24 illustrates the absolute difference between Gaussian curvature of the backlight surface and the equivalent approximation from the triangulation, at the same 3D vertex positions for the Gaussian curvature optimised RATC and diamond configuration triangulations. It can be seen that the diamond configuration had accumulated more points with higher Gaussian curvature differences than the optimised triangulation. By using the optimised triangulation, the average K_G difference per vertex was improved by 37% and 49% for the backlight and windscreen surfaces respectively.

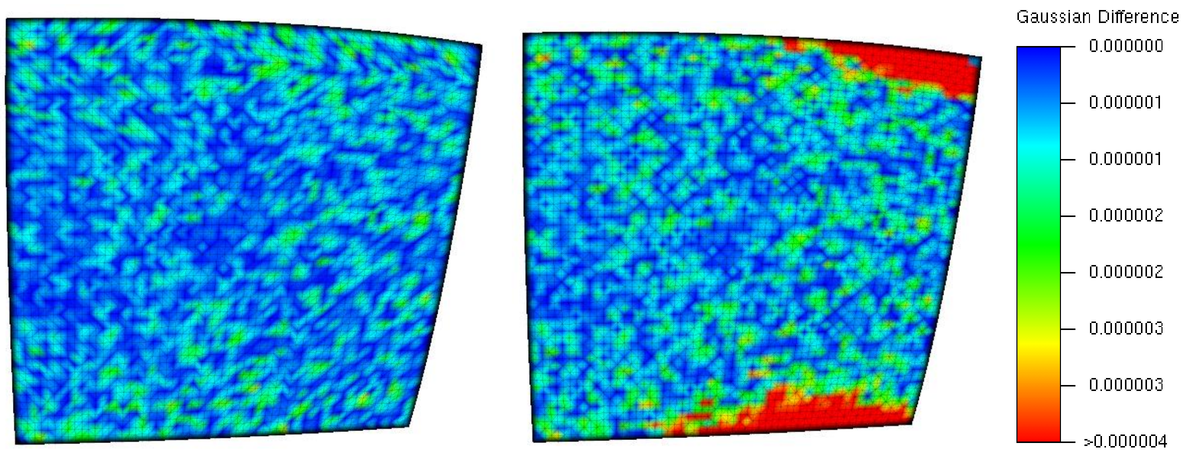


Figure 5.24 Absolute difference per vertex between the surface Gaussian curvature and the (a) Gaussian curvature optimised and (b) diamond RATC.

The relatively poor approximation of Gaussian curvature is attributed to the constant flipping of barycentric area magnitudes between vertices with valency 4 and 8, which are the extreme valencies. Although the improved Gaussian curvature approximation method has greatly enhanced the estimation values, this global configuration still does not compare well against the others. However, a useful feature about the triangulation generated by the Gaussian curvature RATC optimised was that, given a starting point, a well defined triangulation for both point sets was achieved. Therefore, using this method can help remove ambiguity by providing a consistent configuration, which can be useful when comparing different flattening methods.

5.6.3 Flattening Analysis

Tables 5.7 and 5.8 shows that the regular and RATC optimised triangulations generated the best flattening results for the optimum seed triangle. The diamond configurations accumulated the most amount of distortion for the area, followed by the chevron, optimised RATC and regular configurations. This is interesting as it may be interpreted that the triangulation that approximated the Gaussian curvature closest, also generated the best flattening results. Although a better approximation of the surface is no doubt a contributor to the success of the flattening, as illustrated by the results above, it is noted that the seed success rate for the optimised RATC was the lowest in every test. The reason for this low success rate is due to the order in which the triangles were flattened. To illustrate this, the 3 global configurations will be analysed first.

For each global RATC a different mapping order for each triangulation was generated (Figure 5.25), despite using the same edge based mapping order method presented by Cader *et al.* (2005).

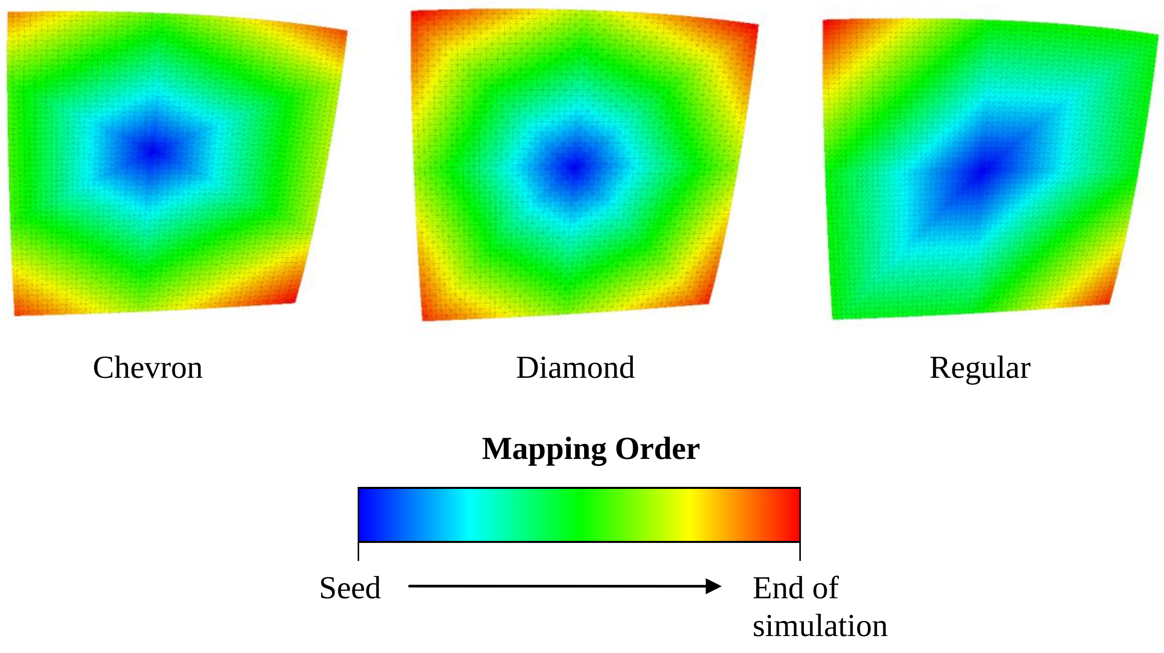


Figure 5.25 Different mapping orders.

This is because for each RATC the triangles can be made up of different vertices, which means that their edges will be incident to different triangles. These topological differences mean that the mapping order will not be the same for each RATC, hence the flattening result may not be the same. Further experimental evidence has shown that out of the 3 global configurations, the regular and chevron configurations give better flattening results and seed success rates, compared to the diamond configuration (Parwana and Cripps, 2009). This is because both of these configurations ensure that every vertex within the triangulation, excluding the boundary edges, is incident to 6 triangles (i.e. valency 6). For these two configurations, the number of triangles and averaged points (AP) processed per level, for the given mapping order are the same (Table 5.9-5.10). It can be seen that both the number of triangles processed and the number of averaged points per level increase linearly, producing a consistent mapping order.

Regular Configuration Mapping Order Data												
Level	Seed	1	2	3	4	5	6	7	8	9	10	11
No. Triangles/Level	1	3	6	9	12	15	18	21	24	27	30	33
Total No. Triangles	1	4	10	19	31	46	64	85	109	136	166	199
No. AP/Level	0	0	0	3	3	6	6	9	9	12	12	15
Total No. AP	0	0	0	3	6	12	18	27	36	48	60	75

Table 5.9 Regular RATC mapping order.

Chevron Configuration Mapping Order Data												
Level	Seed	1	2	3	4	5	6	7	8	9	10	11
No. Triangles/Level	1	3	6	9	12	15	18	21	24	27	30	33
Total No. Triangles	1	4	10	19	31	46	64	85	109	136	166	199
No. AP/Level	0	0	0	3	3	6	6	9	9	12	12	15
Total No. AP	0	0	0	3	6	12	18	27	36	48	60	75

Table 5.10 Chevron RATC mapping order.

Furthermore, for the regular and chevron configurations, the number of triangles, n_t, within the n^{th} level of a mapping order can be defined as:

$$n_t = 1 + 3\sum_{i=0}^{n} i = 1 + 3\left(\frac{n(n+1)}{2}\right) \qquad (5.8)$$

In contrast, the number of triangles and averaged points processed per level in the diamond mapping order is completely different and does not increment linearly (Table 5.11).

Diamond Configuration Mapping Order Data												
Level	Seed	1	2	3	4	5	6	7	8	9	10	11
No. Triangles/Level	1	3	5	8	11	13	16	19	21	24	27	29
Total No. Triangles	1	4	9	17	28	41	57	76	97	121	148	177
No. AP/Level	0	0	1	1	4	5	5	8	9	9	12	13
Total No. AP	0	0	1	2	6	11	16	24	33	42	54	67
No. Duplicate AP/Level	0	0	0	0	0	0	0	3	0	0	3	0

Table 5.11 Diamond RATC mapping order.

Apart from the seed, the number of triangles processed in each level increases in the following pattern: 2, 3, 3, 2 , 3, 3, 2,…. Neglecting the first two levels of the diamond configuration mapping order, the number of averaged points processed per level also changes in a non-linear fashion. These irregular increments are because the valency of the vertices, within this triangulation, varies between 4 and 8 amongst adjacent vertices. These varying valencies also cause an additional problem in that a single vertex can be averaged twice during a flattening simulation (Table 5.11). From level 6, the number of points averaged more than once per level follows the pattern of: 3, 0, 0, 3, 0, 0, 6, 0, 0, 6,…. This duplicate averaging of a single vertex does not occur in the chevron or regular configurations, which will have contributed to the higher seed triangle success rates.

The optimised RATC triangulation ensures that the Gaussian curvature approximation is the best, but at the compromise of the vertex valencies being more irregular across the entire surface. For each of the surfaces, the valencies generated by the optimised RATC triangulation can be seen in Table 5.12.

	Num of vertices with…				
Surface	Valency 4	Valency 5	Valency 6	Valency 7	Valency 8
Backlight	58	103	3007	173	23
Windscreen	41	90	4357	100	36

Table 5.12 Gaussian curvature optimised RATC vertex valencies.

These varying valencies produce far more irregular mapping orders (Figure 5.26), thus contributing to the lower seed success rates. Nevertheless, despite the low success rate, the fact that the triangulation approximates the Gaussian curvature more accurately may have contributed to the minimum distortion of the optimum flattening. It was also noted that the majority of vertices within the optimised RATCs had a valency of 6; minimising the irregularity of the mapping order, which may have helped achieve a good optimum flattening. However, further work would need to be carried out to understand this more rigorously.

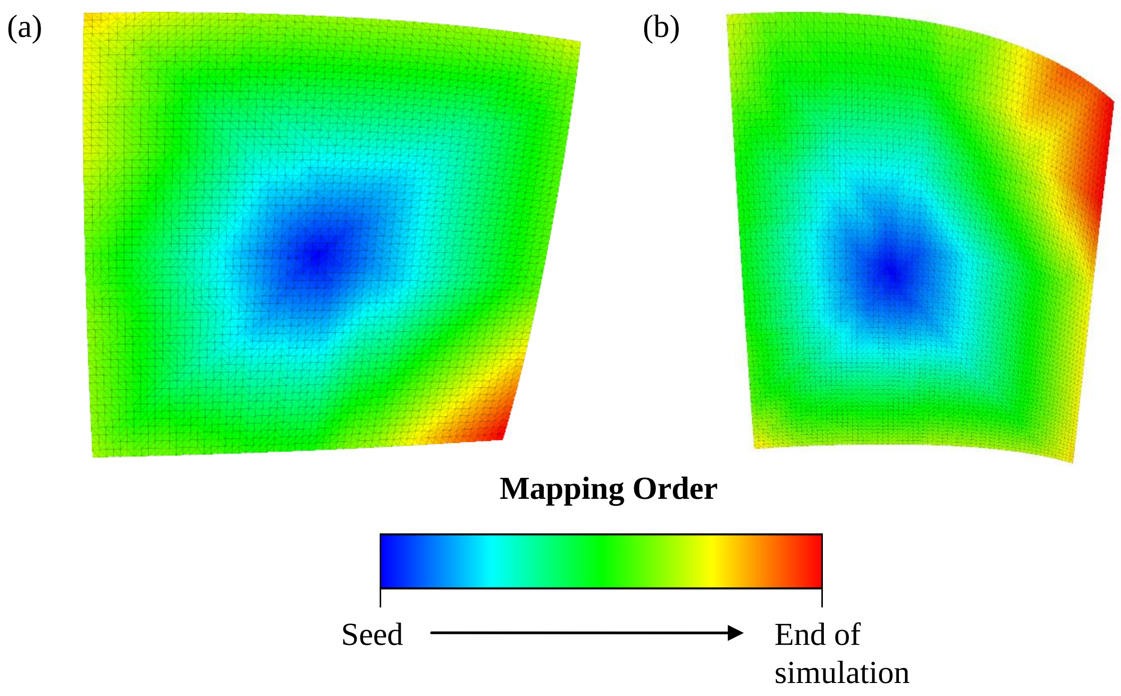


Figure 5.26 Flattening mapping orders for (a) backlight and (b) windscreen triangulations.

The importance of the mapping order is further highlighted by the placement of the optimum seed triangle. This test, and other experimental evidence, has shown that for almost developable surfaces, the optimum seed location is generally towards the centre of the triangulation. By placing a seed closer to the central region of the triangulation, it essentially reduces the number of levels which are created in the mapping order, which in turn, reduces the number of points that are averaged. It also ensures that the distribution of area and shape distortion is spread more evenly through the entire triangulation.

Interestingly, the regular configuration generated the least change in absolute area for both surfaces, but also had the largest absolute shape change. It is possible for two triangles to have different edge lengths but the same area. Therefore, during flattening a triangle's edges may have distorted in a manner such that the 2D triangle generated preserved the original area. The shape of a triangulation is defined by its edges, which are defined by its vertices.

Therefore, sampling points from a surface so that the triangulation inherits the shape of the original surface better could aid in the preservation of shape during flattening. This work is carried out in Chapter 6.

The Gaussian curvature RATC optimised triangulation generated good quality Gaussian curvature approximations and relatively good flattening results. However, the regular configuration appeared to be the best choice of RATC because of its overall success in terms of approximating the Gaussian curvature well, producing the least area change and the highest seed triangle success rate, i.e. algorithm stability. The diamond configuration was the least successful and therefore would not be a good choice for any triangulation being used for flattening.

Chapter 6

TRIANGULATION DEVELOPABILITY

Chapter 5 concentrated on establishing the best connectivity between a given set of points, to achieve the best possible surface flattening. However, in the same way that a triangulation dependent algorithm can only perform as well as its triangulation, a triangulation can only be as good as the underlying points that define it.

When sampling points from the surfaces in Chapter 5, a uniform subdivision method was used, which took no surface characteristics into consideration to decide the point's positional placements. In a topological sense, the points, which are the lowest level of detail, are joined by edges to create an approximation of the original surface structure (i.e. shape). Although an acceptable configuration has been established, the quality of the triangulation may be enhanced further by sampling the points in such a way that the edges which will be created from the points will better reflect the shape of the original surface.

This chapter will investigate changing the triangulation's vertices and the downstream effects on flattening. It will also present a new triangulation method to approximate the original surface's shape better, which will help aid in developing a better flattening result.

There are many different surface characteristics which can be used for determining the positions for points to be sampled. However, the characteristic used in this chapter is the axis of minimum principal curvature (AMPC). To motivate the reason for its usage and how it could preserve the shape of a 3D triangulation during flattening, a novel test performed on a developable surface and its triangulation will be presented first.

6.1 Developable Surface Triangulation Problem

Developable surfaces are ideally suited to surface flattening as they are the only surface definition that will map from 3D to 2D without causing any distortion to the original shape (McCartney *et al.*, 1999; Wang *et al.*, 2002). Although this statement is undoubtedly true, it is not necessarily true for a CAE flattening simulation because at no point is the surface definition directly flattened. The surface is first approximated as a set of triangles, which are then sequentially mapped into 2D. During the triangulation process, the transfer of geometric information from the original surface to the triangulation can get lost, meaning that the triangulation no longer approximates a developable surface. Therefore, the resulting surface flattening will induce distortion, giving an erroneous result.

Consider a developable surface defining part of a cylinder which has zero curvature along constant *v* values as shown in Figure 6.1.

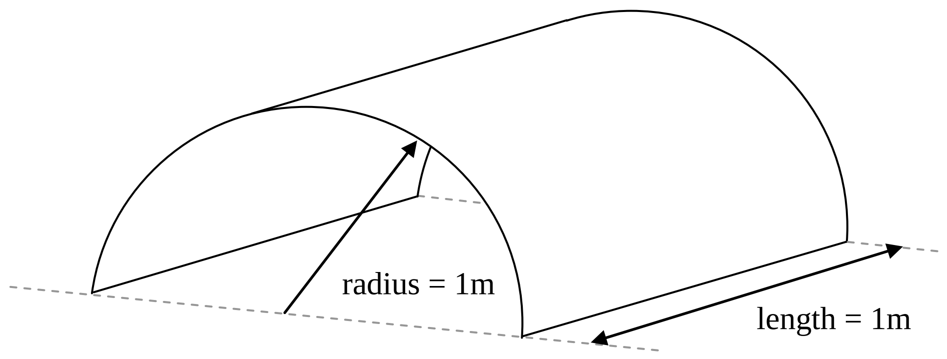


Figure 6.1 Cylinder part surface.

Test 1: Uniformly Subdivided Parameter Domain

This surface was first uniformly subdivided into a grid of 20×20 points, evaluated at equally spaced parametric points (u_i, v_i), which was then triangulated into a regular RATC. This grid density has been chosen for illustration and demonstration purposes, whilst producing a reasonable area approximation (within 5% of the original surface area). In addition, the regular RATC was used within the triangulation because it is widely adopted by many structured triangulation schemes (Rockwood *et al*., 1989; Piegl and Tiller, 1998; Sadoyan *et al*., 2006) and was found to provide the best flattening results in Chapter 5. The triangulation consisted of 772 triangles and was flattened using the Cader *et al*. (2005) algorithm.

To give an unbiased comparison from the results, the optimum seed triangle was determined that resulted in the least change in area after flattening (i.e. distortion). Figure 6.2 shows the original 3D triangulation of the surface and the flattened 2D triangulation. Figure 6.2 also shows the percentage absolute change in area, ΔA (4.9), and shape, ΔS (4.10), between the 3D triangulation and the 2D flattening (Figure 6.2(a)) and both the minimum ($K_{GT}min$) and maximum ($K_{GT}max$) Gaussian curvature approximations derived from the 3D triangulation using (5.7) (Figure 6.2(b)). The line of zero principal curvature (LZPC), which is parallel to the AMPC is also shown on the 3D triangulation. It is noted that all the strings of sampled points, in the *u* and *v* directions, lie parallel to the axis of principal curvatures.

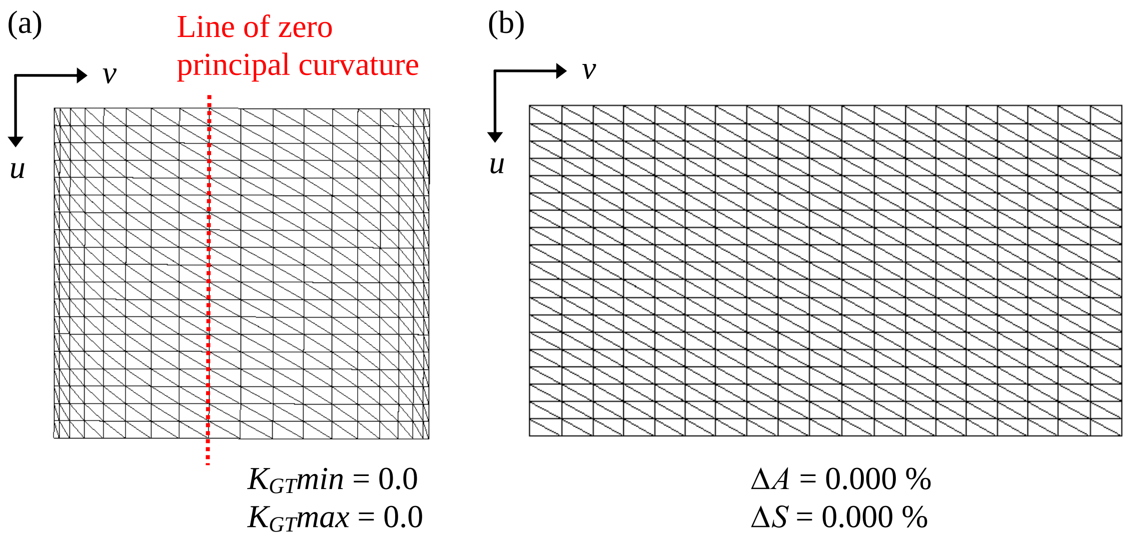


Figure 6.2 (a) 3D triangulation of part cylinder. (b) Flattened triangulation.

As would be expected, no deformation was induced when this triangulation was mapped into 2D, within tolerances of 1mm and 1mm^2 for ΔA and ΔS respectively. In addition, the Gaussian curvature approximated by the triangulation was zero over of entire surface.

Test 2: Points Skewed in the *u* Parameter Direction

The surface was re-sampled using the same grid size. This time, points in the *u* direction were skewed by distorting the parameter spacing along one boundary and linearly blending to the opposite fixed boundary (Figure 6.3(a)). The amount of skew, which is applied to a single boundary point, is defined as shown in Figure 6.3(b), where 0% is no skew and 100% is moving the current point fully to the position of next boundary point along that boundary edge. Experimental evidence has shown that increasing the percentage of skew has the effect of increasing the amount of distortion induced within a triangulation.

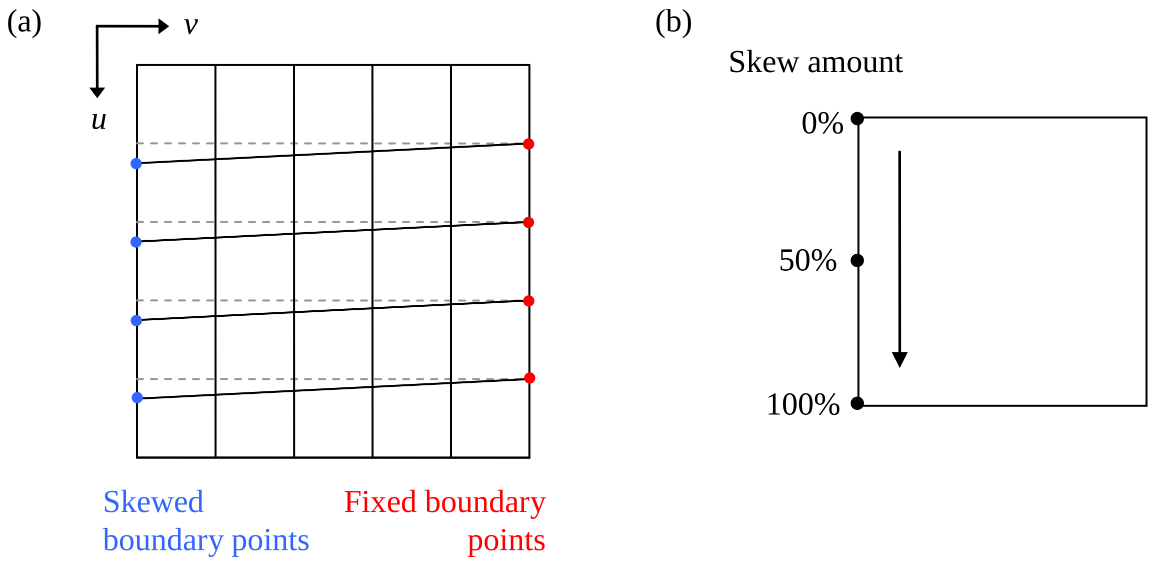


Figure 6.3 (a) Skewed points in the *u* parameter direction. (b) Definition of skew amount.

Therefore, a skew of 90% was applied to the point sampling to help illustrate the effect of skewing the parameterisation in the *u* parameter. The points in the *v* direction remained fixed, following constant values of *v* (Figure 6.4(a)).

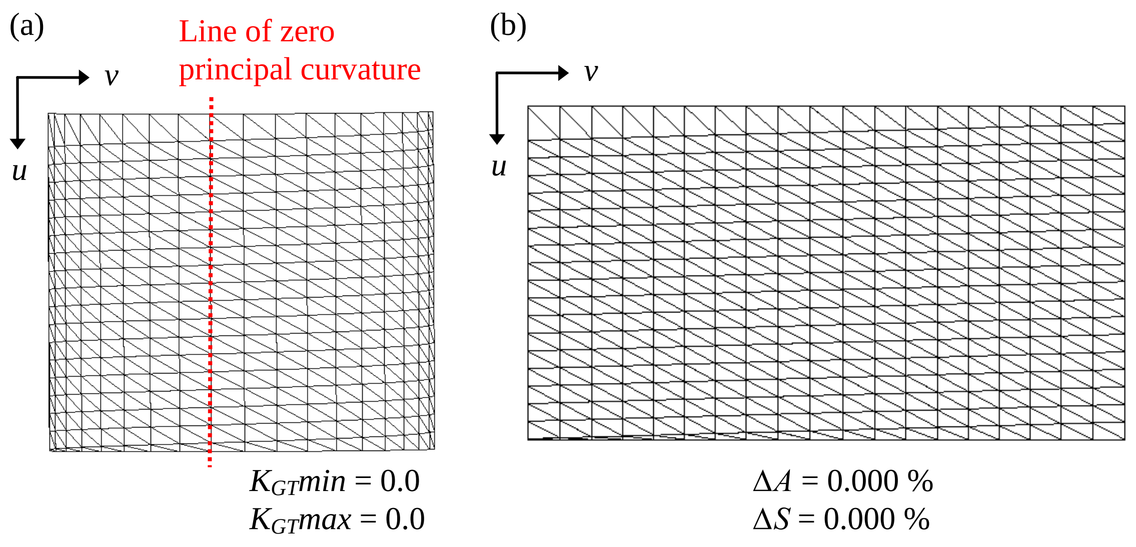


Figure 6.4 (a) 3D triangulation of part cylinder. (b) Flattened triangulation.

Despite the skew imposed in the *u* parameter direction on the triangulation, the strings of points in the v direction continued to follow along the LZPC. Therefore, at least two points (i.e. one edge) of every triangle followed the LZPC, meaning that the triangulation topology inherited enough information from the original surface to approximate a developable surface (Parwana and Cripps, 2009) and flatten without distortion, to within tolerance (Figure 6.4(b)).

Test 3: Points Skewed in the *v* Parameter Direction

A third grid of points was sampled where the *u* values were kept constant and the *v* values were skewed by 90%, using the same approach as before. In this case, the *v* parameter values were no longer parallel to the LZPC (i.e. AMPC) (Figure 6.5(a)). Further, no string of points followed the LZPC meaning that the topological information of the triangulation no longer represents a developable surface. This resulted in a distorted flattening of the triangulation, i.e. greater than tolerance (Figure 6.5(b)).

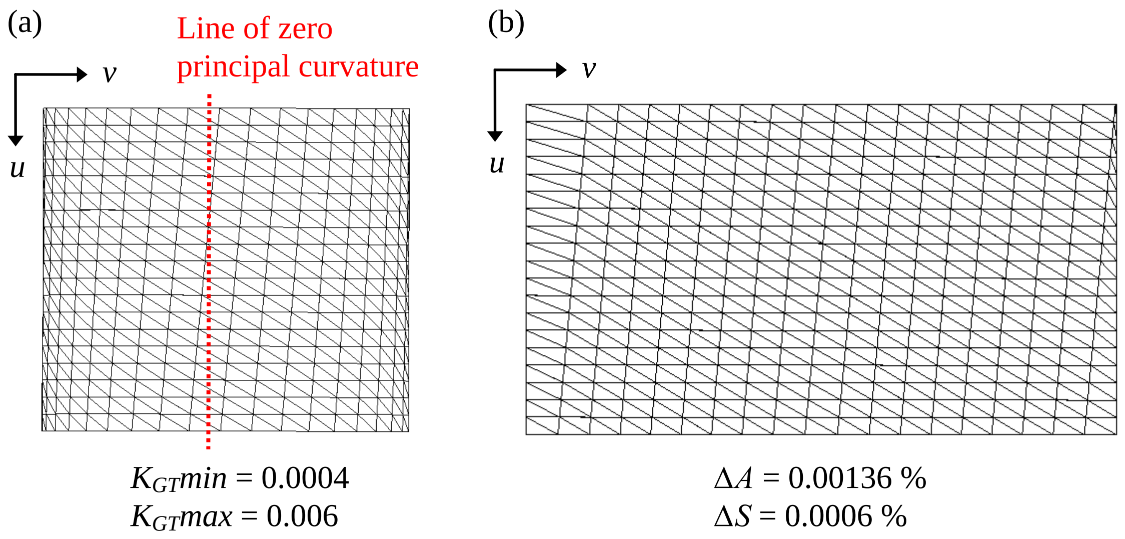


Figure 6.5 (a) 3D triangulation of part cylinder. (b) Flattened triangulation.

Figure 6.6 shows the 2D flattening and the magnitudes of positional movement for each flattened vertex, i.e. the difference between its original flattened position and its final position after any runtime modifications (vertex averaging as described in Section 4.2.1).

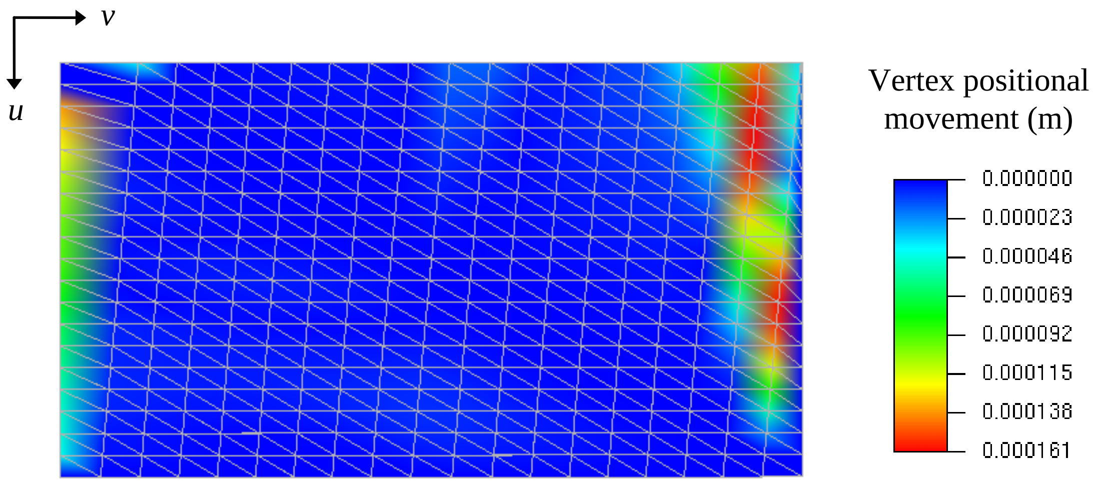


Figure 6.6 Flattened triangulation showing vertex positional movement.

Figure 6.7 shows the 2D flattening and the area difference between the 3D triangles and their corresponding flattened triangles. It can be seen that the regions of the flattened model that have vertex positional movement are the same regions that have an area change.

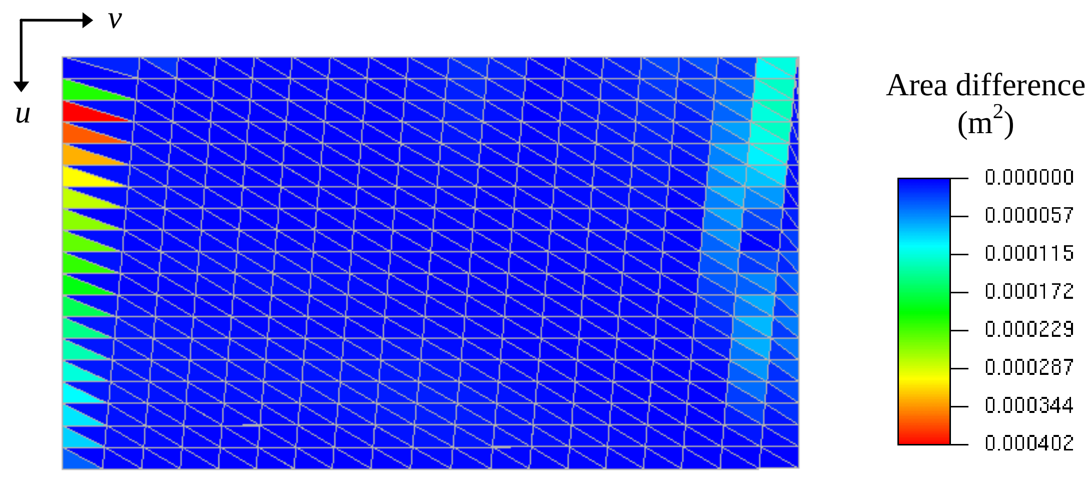


Figure 6.7 Flattened triangulation showing triangle area difference.

The distortion caused by this 3D triangulation is an unexpected result for a developable surface, since such a surface should flatten without any vertex positional movement or triangle area change (McCartney *et al*., 1999; Wang *et al*., 2002). It is noted that whilst ΔA and ΔS are small, i.e. 0.00136% and 0.006% respectively (Figures 6.6 and 6.7), the flattening results shown above were generated by the optimum seed triangle (Section 4.4.4). Therefore, when flattening this 3D triangulation with other seed triangles, larger magnitudes for area and shape changes were generated with the maximum ΔA and ΔS being 0.02% and 0.025% respectively. Furthermore, the method used to skew the parameterisation kept all points along 3 boundary edges fixed and used a linear blending method, thus maintaining a well defined triangulation. Increasing the severity of the skewing, in the *v* parameter direction (i.e. direction of zero curvature), can cause the results from a flattening simulation to get worse and even fail (Parwana and Cripps, 2009). This is illustrated in the following example.

The cylinder part was subdivided again into a grid of 20×20 points and triangulated into 722 triangles. Before triangulating points, a linear skew of 90% was applied in both the *u* and *v* directions. To make the points placement more erratic, the skewing was only applying to points $\mathbf{p}_{i,j}$, for *i*,*j*=1, 3, …, 19, where *i* and *j* are the vertex indices in the *u* and *v* directions respectively. This would cause the grid of points to oscillate and be more irregular as shown in Figure 6.8.

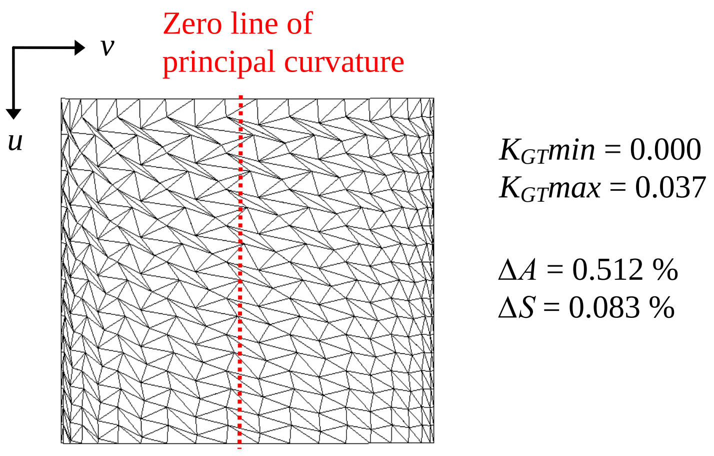


Figure 6.8 Skew severity increased.

In this case, the maximum ΔA and ΔS generated from the flattening was 9.26% and 2.83% respectively. Furthermore, out of the 722 simulations performed on this triangulation, only 375 (i.e. 52%) flattened without failure. As in Figure 6.5, the strings of vertices for the triangulation in Figure 6.8 do not follow along the LZPC. In addition, as the severity of the skewing in the latter's triangulation has been increased, it has inherited less developability from the surface. This is evident from K_{GTmax}, which has increased by a factor of 6 from the triangulation shown in Figure 6.5.

The increased skewing has also produced many long thin triangles in the triangulation. An explanation for the 48% seed triangle failure rate could be due to these long thin triangles, as they are numerically unstable and can cause triangle orientation flipping during averaging (Section 4.4.2). However, this may not be the case as experimental evidence has suggested that triangles with high aspect ratios do not induce deformation or algorithm failure for developable surfaces. This is because, multiple 2D flattened positions for a single 3D point will be coincident, within a numerical tolerance, for developable surfaces. Therefore, the averaging process will not cause any distortion to be induced. Therefore, as the triangulation shown in Figure 6.8 is approximating a surface with non zero Gaussian curvatures (i.e. non-developable), it is possible for distortion and triangle orientation flipping to occur during vertex averaging.

6.2 Minimum Principal Curvature Line Hypothesis

It was shown in Section 6.1 that the triangulation of a developable surface may not necessarily be developable. To ensure developabilty was inherited from a surface, it is suggested that should be sampled along the LZPC. This would ensure that every triangle would have one edge that followed the AMPC, meaning that the triangulation's shape (i.e. edges) would more accurately approximate the original developable surface's structure.

For non-developable surfaces, experimental evidence has shown that surfaces with larger Gaussian curvatures induce larger amounts of area and shape distortion into the final flattening result. However, as the surfaces being used throughout this thesis are almost developable, it is suggested that aligning the sampled points along the AMPC would generate a triangulation that would inherit the characteristic of developability better. Inheriting more developability in the triangulation's shape would have the effect of minimising the amount of shape distortion, which in turn should induce less area distortion into a flattening simulation.

Currently, no methods exist for generating a triangulation from a surface definition by sampling points along the axis of minimum (or maximum) principal curvature. Therefore, a new algorithm is proposed which will subdivide the parameter domain of a developable or almost developable surface, where the direction of the point sampling will be influenced by the AMPC. The points being sampled must be generated in such a way that they create cells, which can then be triangulated by adding a diagonal, i.e. simulating a structured triangulation.

6.3 Algorithm Overview

This section will provide an overview of the algorithm to subdivide and triangulate the parameter domain a surface, which is influenced by the AMPC. Full details of each stage will be provided in Section 6.4.

6.3.1 Minimum Principal Curvature Direction Strips

The method begins by uniformly subdividing the parameter domain into $m \times n$ points, where *m* and *n* are determined so that the area approximation of the triangulation is the same as the original area within a numerical tolerance. These points will then be used to establish the modal minimum principal curvature direction (MPCD) (i.e. *u* direction or *v* direction) (Section 6.4, Stage 2). As almost developable surfaces are being used, the MCPD will generally follow one direction consistently. Therefore, the most occurring (i.e. modal) MCPD is used.

Once calculated, the parameter domain will be subdivided into *r* strips of cells, which follow the modal MPCD (Figure 6.10). Details of determining *r* will be provided in Section 6.4, Stage 3. The remainder of this overview will assume the strips follow the *v* direction. The approach for the u direction is analogous.

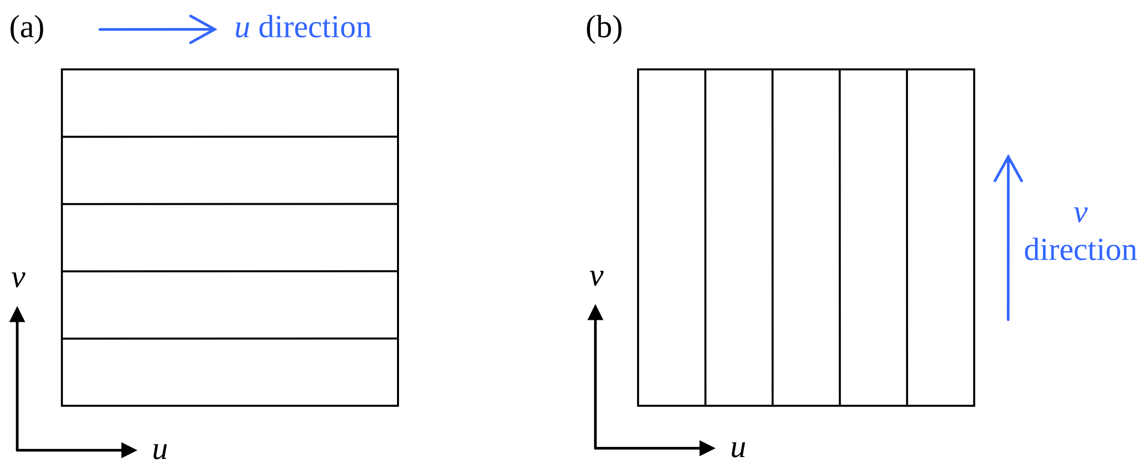


Figure 6.10 Subdividing parameter domain *r* strips for (a) *u* and (b) *v* modal minimum principal curvature directions.

6.3.2 Curvature Polylines and Triangulation

Each strip is then individually operated on to find a curvature polyline, which follows the AMPC in the modal MPCD (Section 6.4, Stage 4). The curvature polyline must be fully contained within each strip, which will ensure that no two curvature polylines intersect (Figure 6.11). It is noted that, in some situations, a strip may not have a curvature polyline that is fully contained within it. In this case, these strips will remain empty after this step (Figure 6.11).

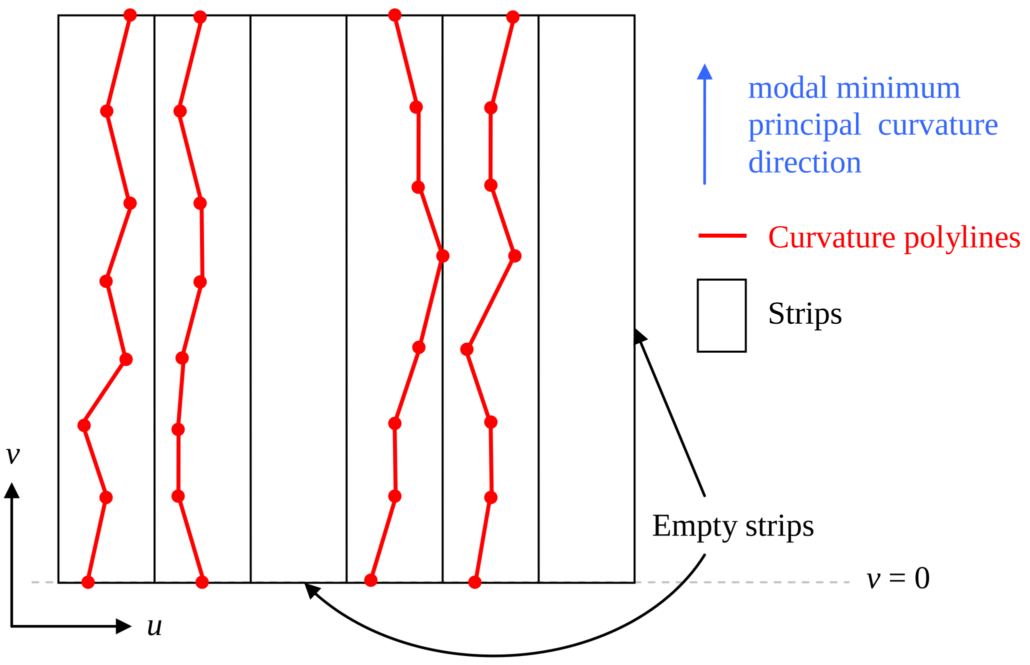


Figure 6.11 Generating curvature polylines for each parameter domain strips.

The next stage is to generate intermediate curvature polylines to ensure that the distance between adjacent points along $v = 0$ ($u = 0$ for u modal MPCD) is no more than the step size $1/m$ ($1/n$ for u direction). Therefore, the number of intermediate curvature polylines, *np*, needed to meet this step size requirement is calculated (Section 6.4, Stage 6) and then the new polylines are inserted by linearly blending between the two polylines being operated on (Figure 6.12).

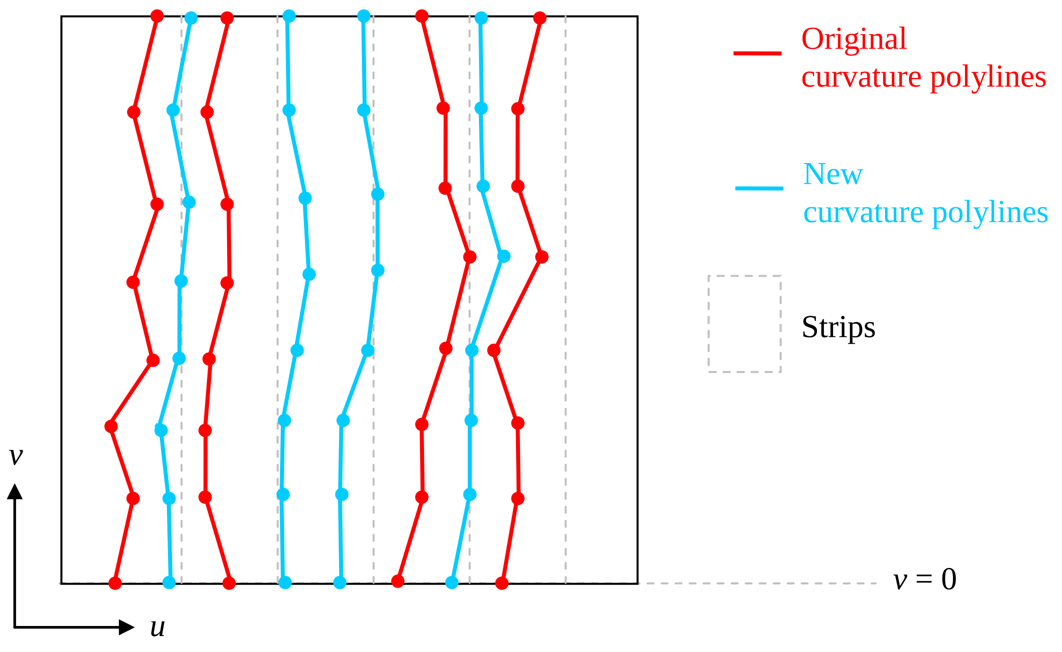


Figure 6.12 Generated new curvature lines to meet the step size requirement.

The curvature polylines can then be converted into a grid of cells, which are then triangulated by adding a diagonal to each cell (Figure 6.13).

6.3.3 Boundary Edge Curvature Polylines and Triangulation

After this initial triangulation, parts of the parameter domain located outside of the first and last curvature polylines may still remain un-triangulated (Figure 6.13). Therefore, new points will need to be generated to fill these un-triangulated regions by offsetting the outer curvature polylines outwards, by a step size of $1/m$ ($1/n$ for u direction). This process will be performed on each of the un-triangulated regions (Figure 6.13) until a curvature polyline is generated where all of the points are located outside of the parameter domain (Figure 6.14). The points of these offset curvature polylines are then revisited and modified by clamping any polyline points outside the parameter domain to the nearest vertical (horizontal for u direction) parameter domain edge (i.e. u = 0, u = 1) (Figure 6.15). These offset points are then converted into cells and triangulated in the same manner as the interior points, with an additional check to see whether the cell points are all collinear, i.e. the u coordinates of all points are the same, within tolerance (v coordinate for u direction). If this occurs, the cell is collinear (i.e. degenerate) and is not triangulated (Section 6.4, Stage 8).

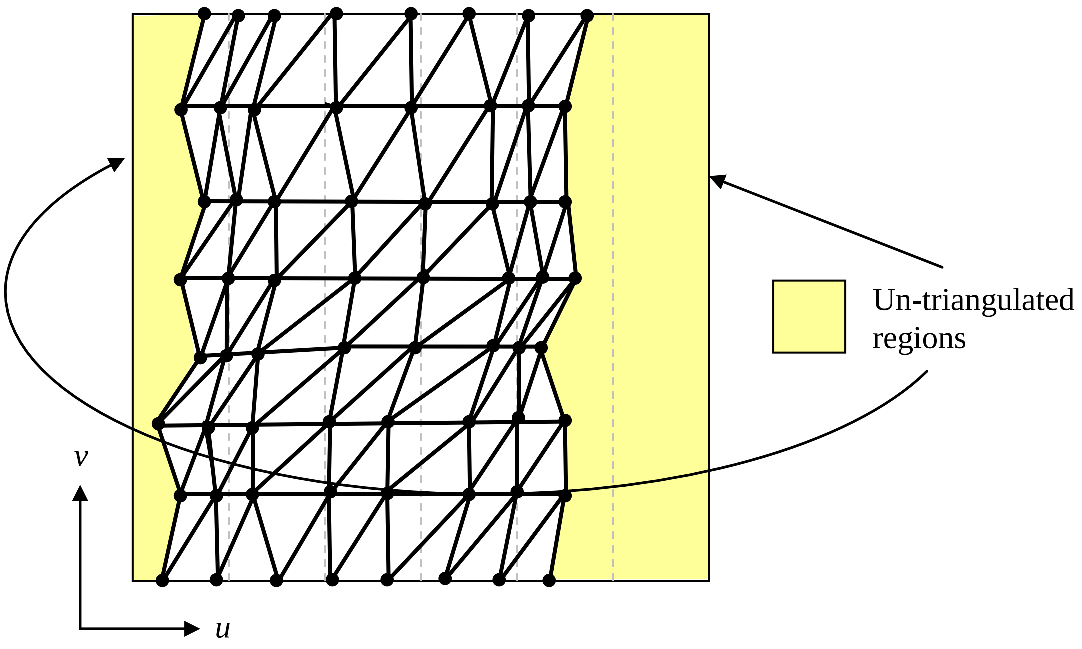
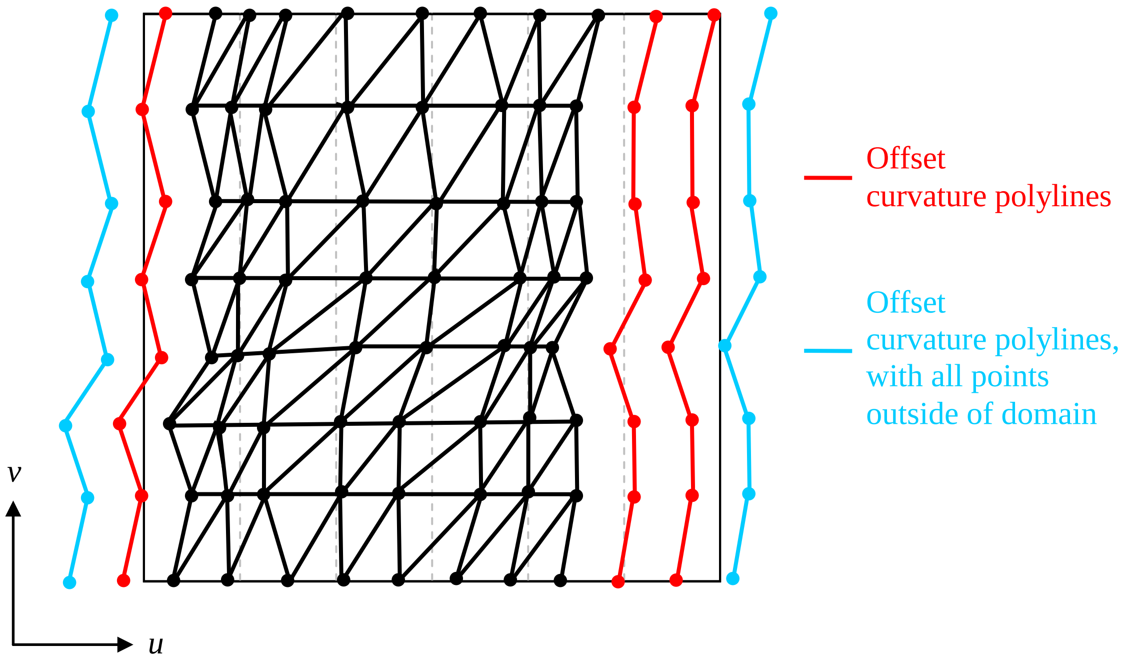


Figure 6.13 Triangulated and un-triangulated regions of parameter domain.



Offset curvature polylines
Offset curvature polylines, with all points outside of domain
v
u

Figure 6.14 Offset curvature polylines generated.

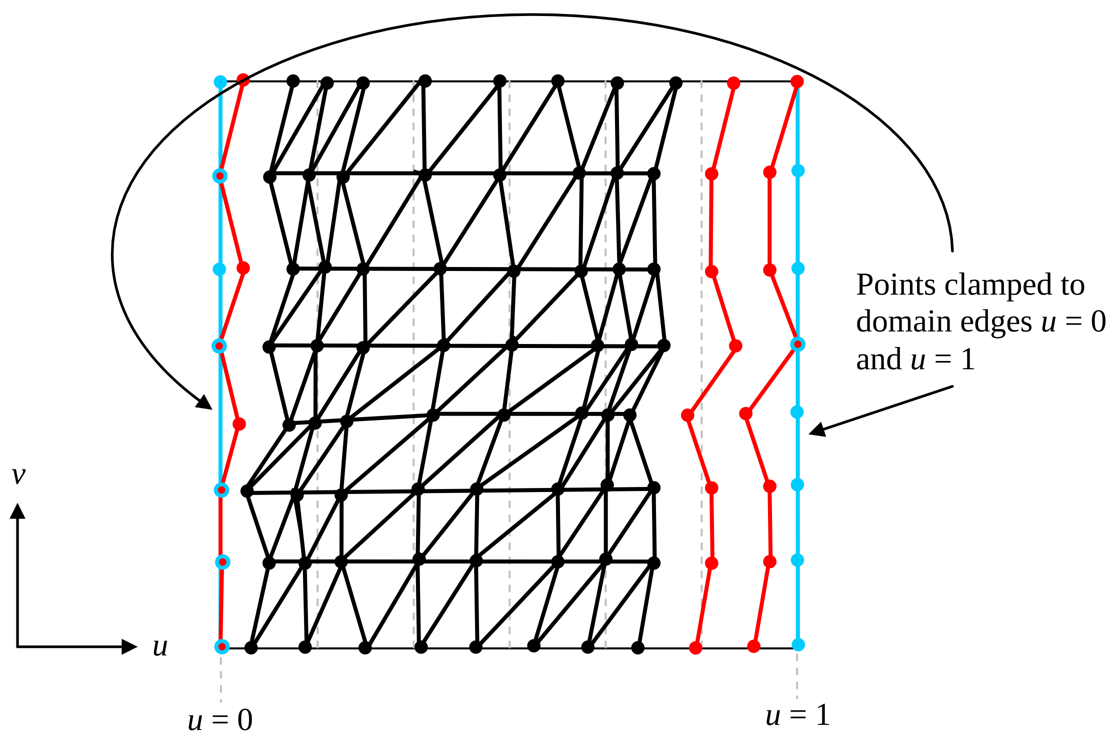


Figure 6.15 Clamping points outside of the parameter domain to the nearest edge.

6.4 Algorithm Details

This section will provide full details of the different algorithm stages outlined in Section 6.3.

Stage 1: Subdividing the Surface

Given a NUBS surface, $\mathbf{S}(u, v)$, defined over the normalised parameter domain $u,v \in [0,1]$ (Section 1.1.3), the first stage is to represent the parameter plane as a uniformly subdivided $m \times n$ grid of points. m and n are user defined quantities, which define the mesh density. In this thesis, m and n were chosen so that the approximation of the triangulation area was within a tolerance of 2mm^2 of the original surface area.

Stage 2: Establish Minimum Principal Curvature Direction

The next stage is to evaluate the surface's principal curvatures and directions at each grid point using the method shown in Section 1.4.2. At each of these points, the minimum curvature κ_{min} and its direction ($du_{\min}$, $dv_{\min}$) is checked to establish which parameter direction the minimum curvature line is following. If $|du_{\min}| > |dv_{\min}|$, then the AMPC tends towards the u (horizontal) direction, otherwise if $|du_{\min}| < |dv_{\min}|$, then it is tending towards the v (vertical) direction. If the two directions are equal, within a zero tolerance, then no direction is recorded. After evaluating the surface at all the grid points, the modal direction will be taken to be the MPCD. If there were no directions recorded, during this testing phase, or the modal values for each direction are the same, then both directions have the same influence in which case either the u or v direction can be used.

The remainder of this algorithm will assume that the MPCD tends towards the vertical direction (v). The method used for the horizontal direction (u) is analogous.

Stage 3: Splitting the Surface into Regions

The next stage is to subdivide the parameter domain into strips of rectangular regions along the minimum principal curvature direction. For the vertical direction, the parameter domain will be split into $r = m/2$ strips ($n/2$ strips for horizontal direction) as shown in Figure 6.16(a).

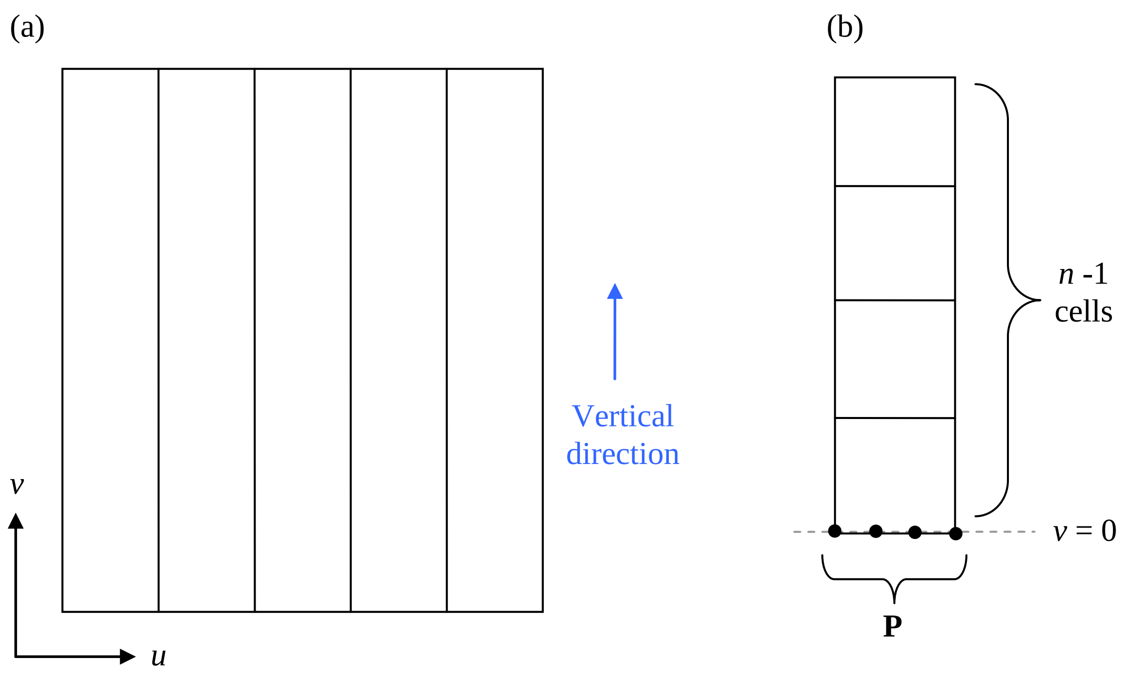


Figure 6.16 (a) Subdividing parameter domain vertically into *m*/2 strips. (b) Generating cells from a strips and subdividing *v*=0 into num_s points.

The next stage is to take each strip and subdivide the *v* direction into *n*-1 cells (*m*-1 for horizontal direction) and then partition the bottom edge of the cell (i.e. along *v*=0 edge) into a set of *s* points $\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{s-1}\}$ (Figure 6.16(b)). The number for subdivisions, *s*, is user specified where a larger number will generate a greater number of curvature polylines for a single strip. This will have the effect of performing more comparisons between curvature polylines to find one which has the least horizontal range (Stage 4). Larger values of *s* will make a curvature influence more accurate, but at the cost of longer processing times. Experimental evidence has shown that a value between 5 and 10 is more than sufficient for almost developable surfaces.

Stage 4: Generating Polylines of Minimum Curvature

The generation of the minimum curvature polyline begins by taking the first point, $\mathbf{p}_0(u_0, v_0)$, and calculating the MPCD, from the actual surface. This direction, (du_0, dv_0) (Figure 6.17(a)), is used to generate a straight line, $\mathbf{l}_0$, which is bounded between $v = v_0$ and $v = v_0 + 1/(n - 1)$, has the gradient dv_0/du_0 and passes through $\mathbf{p}_0$. A second line, $\mathbf{l}_1 = v0 + 1/(n - 1)$, is generated which is parallel to the u axis. The intersection point, $\mathbf{p}_{int}$, between $\mathbf{l}_0$ and $\mathbf{l}_1$ is calculated and then joined with $\mathbf{p}_0$ to form the first polyline segment of the principal curvature line (Figure 6.17(b)).

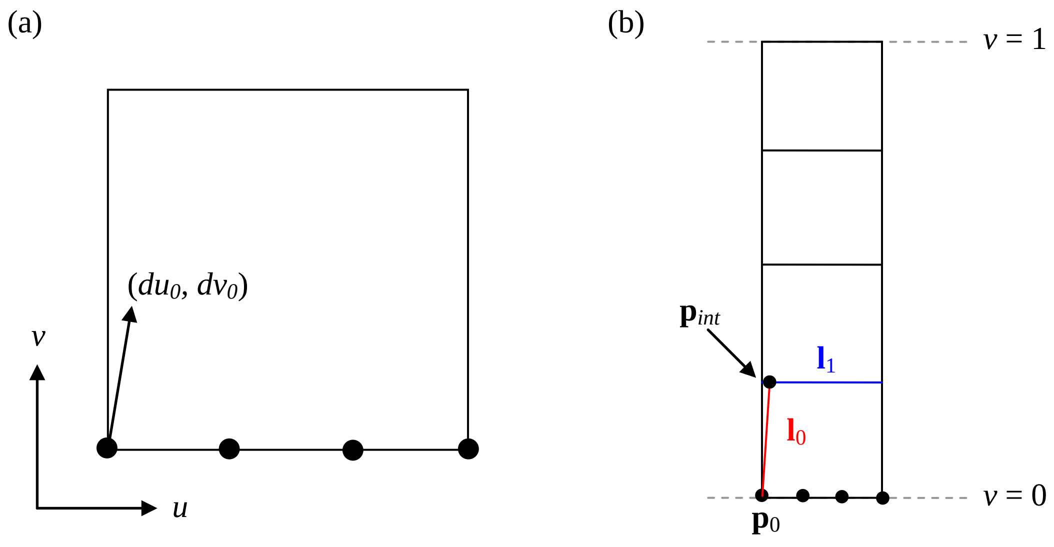


Figure 6.17 (a) Vertical principal curvature direction at $\mathbf{p}_0$ and (b) intersection between $\mathbf{l}_0$ and $\mathbf{l}_1$.

To complete the minimum principal curvature polyline, this process is repeated by evaluating each $\mathbf{p}_{int}$ for its MPCD. This iterative process is continued until the cell edge v=1 is reached, which will terminate the current polyline (Figure 6.18). The minimum polyline $\mathbf{pl}_{min}$, for this strip is stored. The next minimum curvature polyline will be then be generated by starting the process from then next point in the set $\mathbf{P}$.

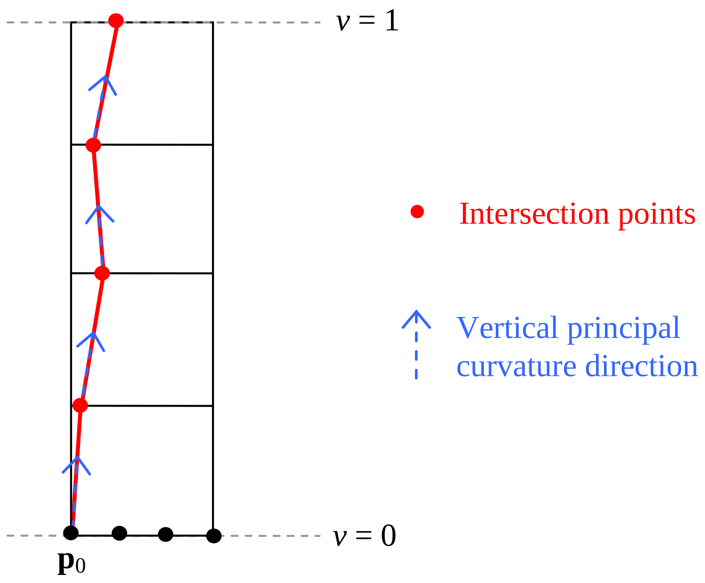


Figure 6.18 Vertical curvature polyline generated from $\mathbf{p}_0$.

Each time a new curvature polyline is created, its horizontal range will be checked against the current $\mathbf{pl}_{min}$ (Figure 6.19). If the current curvature polyline has a smaller horizontal range than $\mathbf{pl}_{min}$, then this current polyline will be stored as $\mathbf{pl}_{min}$, otherwise it will be rejected. This will ensure that the straightest minimum curvature polyline is retained for each strip.

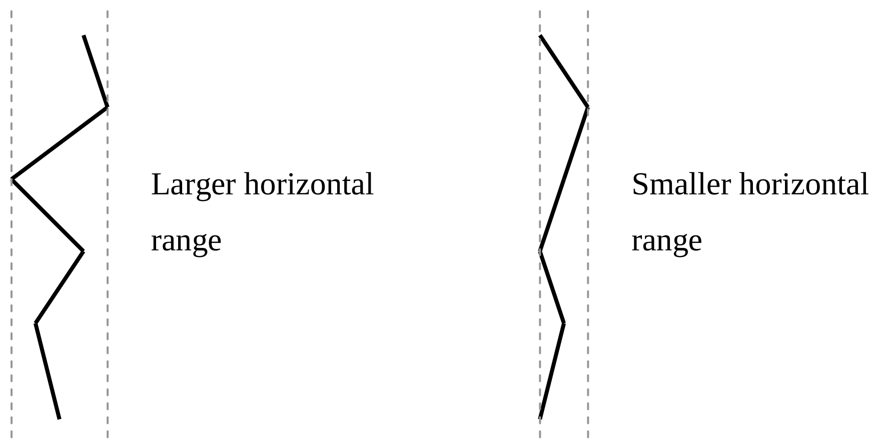


Figure 6.19 Comparing two horizontal ranges of curvature polylines.

By fully containing each minimum curvature polyline within a strip, no adjacent polylines will intersect. However, when generating the polylines, in some cases, lines $\mathbf{l}_0$ and $\mathbf{l}_1$ may not intersect. This will mean that $\mathbf{l}_0$ will have an end point outside of the strip boundaries. As the polyline is no longer fully contained within a strip, this could cause intersection with adjacent polylines. Therefore, if $\mathbf{l}_0$ and $\mathbf{l}_1$ do not intersect, the current polyline is deemed invalid and any previous segments contributing to it are rejected. The process is then restarted using the next point in the set **P**. As a consequence of this check, it may be possible for a strip not to have a $\mathbf{pl}_{min}$. This is not a problem, as these empty strips are filled by strategically creating new curvature polylines as shown in Stage 6.

Stage 5: Checking Minimum Curvature Polylines

Once all the strips have been processed, if no $\mathbf{pl}_{min}$ polylines were generated for any strips, this implies that the strips were too narrow and therefore must be altered to generate some minimum curvature polylines for this surface. Therefore, the method returns to Stage 3 and subdivides the parameter domain into r - 1 strips, before Stages 4 and 5 are repeated. This process is continued until at least one minimum curvature polyline is generated.

Stage 6: Linearly Blending Between Minimum Curvature Polylines

The $\mathbf{pl}_{min}$ polylines generated may have gaps between them which exceed the required u direction step size along $v = 0$. Therefore, new curvature polylines may need to be generated between existing pairs to ensure this step size criterion is met. The number of polylines, n_p, needed in between consecutive pairs of polylines, $\mathbf{pl}_i$ and $\mathbf{pl}_j$, is determined by taking the first point of each polyline, $\mathbf{p}_i(u_i, v_i)$ and $\mathbf{p}_j(u_j, v_j)$ (i.e. points along the v=0 boundary) and using:

$$n_p = \lceil (u_j - u_i)m \rceil - 2 \tag{6.1}$$

For minimum curvature polylines defined in the u direction, the number of intermediate polylines needed is:

$$n_p = \lceil (v_j - v_i) n \rceil - 2 \qquad (6.2)$$

The new polylines, $\mathbf{pl}_k$, k=0, 1, …, n_p-1, are generated by linearly blending between $\mathbf{pl}_i$ and $\mathbf{pl}_j$, to generate the required polylines.

Stage 7: Triangulation Minimum Curvature Polylines

Once a full set of minimum curvature polylines has been generated, they will represent a structured grid of non-intersecting point-strings which can be converted into cells. These cells will then be triangulated by adding a diagonal, i.e. regular configuration. This regular configuration has been chosen as it was found to give the best flattening results in Chapter 5 (Section 5.6.3). It will also allow for a direct comparison between a regular RATC generated from a uniformly sampled grid, and one which has been generated by following the 'ideal' AMPC. The flattening results for each method will be presented and compared in Sections 6.5 and 6.6.

Stage 8: Triangulating Boundary Edges

After triangulating between the minimum curvature polylines, it possible that parts of the surface, located on either side of the current triangulation, may still need to be triangulated (Figure 6.13).

To triangulate these parts of the surfaces, new points will be generated by offsetting the outer minimum curvature polylines by a set size of 1/*m* (1/*n* for *u* direction). The offsetting of the polylines is continued until a polyline is generated, where all the points are located outside the parameter domain. The offset polylines are then revisited, and any points that are contained outside of the parameter domain are clamped to the nearest vertical parameter domain edge (horizontal edge for *u* direction).

The final stage is to generate cells between pairs of offset polylines, and triangulate them by adding a diagonal. Two checks must be carried out when creating the cell to ensure that they will form non-degenerate triangles. First, if all the points of a cell are collinear, with tolerance**,** (i.e. all have the same *u* coordinate), then the cell is degenerate and not triangulated. Secondly, if any two points of the cell are coincident, within a zero tolerance, then the duplicate point is removed and the 3 remaining points are used to generate a triangle (Figure 6.20(b)).

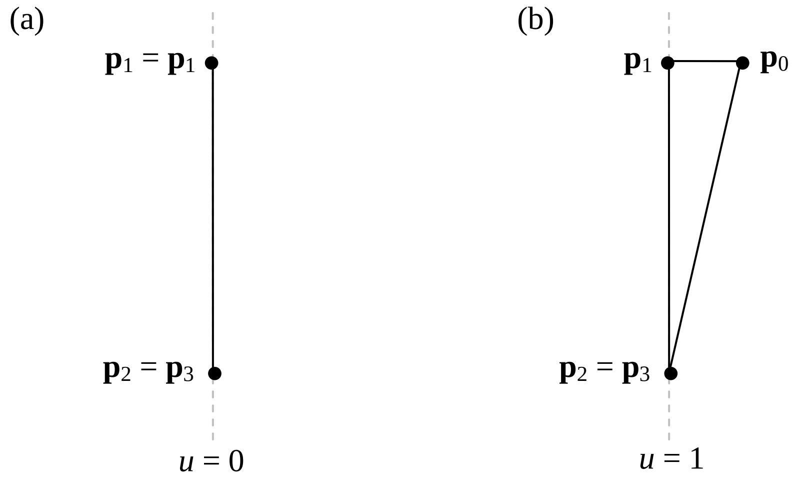

(a)
$\mathbf{p}_1 = \mathbf{p}_1$
$\mathbf{p}_2 = \mathbf{p}_3$
$u = 0$
(b)
$\mathbf{p}_1$
$\mathbf{p}_0$
$\mathbf{p}_2 = \mathbf{p}_3$
$u = 1$

Figure 6.20 (a) Cell points all collinear. (b) 2 cell points are coincident.

6.5 Principal Axis Triangulation Testing

This section will analyse the quality of the flattening results generated by a triangulation that attempts to align the points along the AMPC. The backlight and windscreen surfaces will be used again as the test pieces. A comparison will also be conducted between the flattening results generated from this triangulation and the uniformly sampled regular RATC triangulations from Section 5.6.1.

Each triangulation will be put through multiple flattening simulations to find the optimum seed triangle that induces the least absolute area (ΔA) distortion. In addition, the Gaussian curvature of the triangulation will also be analysed to check the quality of its approximation.

6.5.1 Results

To ensure the surface area approximated by the 3D triangulation was within a 2mm^2 tolerance of the original area, the backlight and windscreen surfaces were subdivided into 62×62 and 70×70 points respectively (Figure 6.21).

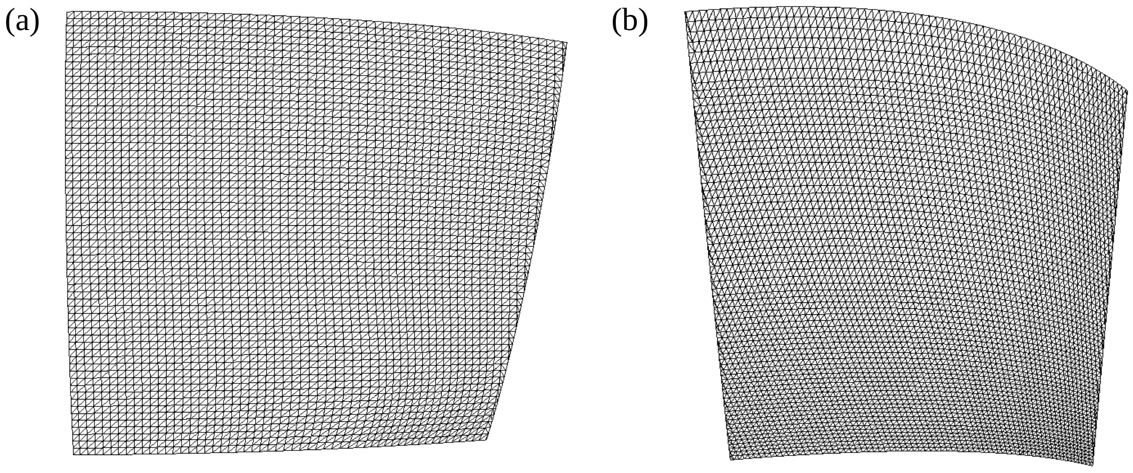


Figure 6.21 (a) Backlight and (b) windscreen surface 3D triangulations.

For almost developable surfaces, in general, the minimum principal curvature lines generated by this triangulation algorithm will not necessarily follow along the iso-parameter lines of the surface. Therefore, many long thin triangles will be formed along the boundary edges of the surface. Figure 6.22 shows the long thin triangles generated along the boundary edges for the windscreen surface. These long thin triangles could cause algorithm instability during flattening (Section 4.4).

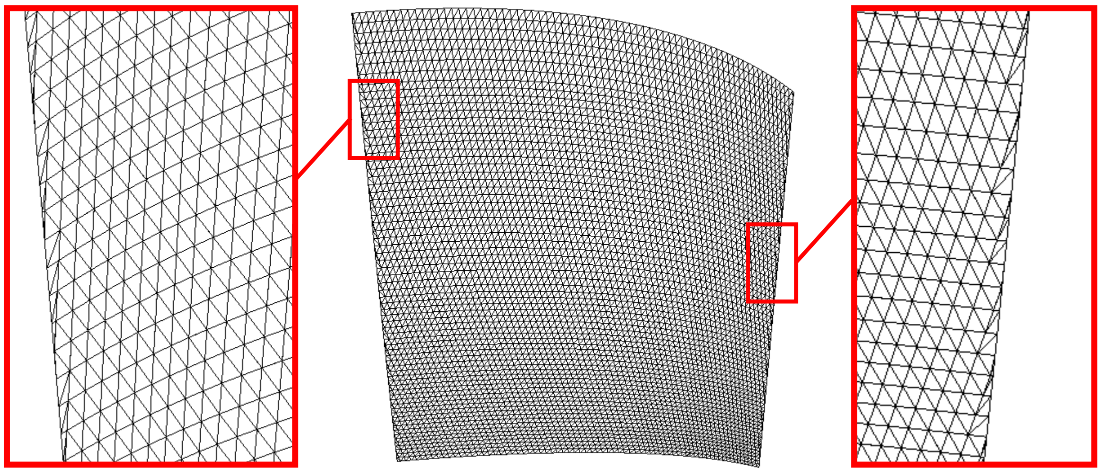


Figure 6.22 Long thin triangles along the boundaries of the windscreen surface triangulation.

Therefore, to make the flattening algorithm more stable, these long thin triangles were removed (Figure 6.23). This was done during the boundary edge triangulation process (Section 6.4, Stage 8) by only triangulating cells where the distance between every pair of points was more than 1mm. Removing these long thin boundary edge triangles is completely feasible for trimmed surfaces, where the un-triangulated cells are outside of the trimming region (Section 4.4.3).

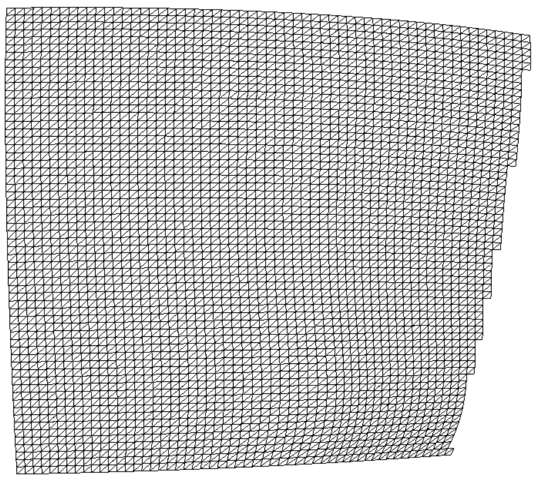
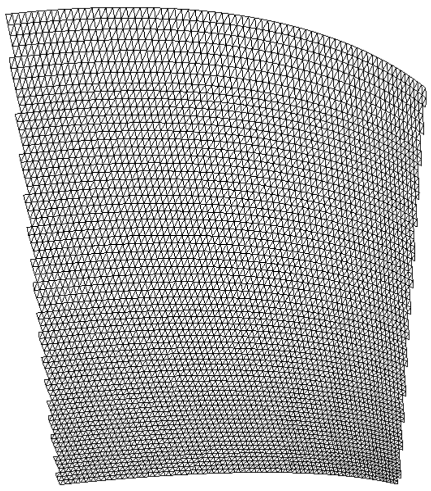
(a)  (b) 

Figure 6.23 (a) Backlight and (b) windscreen surface 3D triangulations.

The triangulation data for each surface definition can be seen in Table 6.1.

Surface	No. Vertices	No. Edges	No. Triangles
Backlight	3542	10381	6840
Windscreen	4901	14392	9492

Table 6.1 Topological data.

The minimum, K_{Gmin}, and maximum, K_{Gmax}, Gaussian curvatures evaluated from the surface sampled points are shown in Table 6.2.

Surface	K_{Gmin}	K_{Gmax}
Backlight	0.000022	0.000098
Windscreen	0.000002	0.000088

Table 6.2 Surface Gaussian Curvature.

Table 6.3 shows the Gaussian curvature approximations generated from each triangulation for the backlight and windscreen surface.

Configuration	K_{Gmin}	K_{Gmax}	Min K_G diff	Max K_G diff	K_G diff per vertex
Backlight	0.000021	0.000098	0	0.000004	0.00000074
Windscreen	0.000002	0.000091	0	0.000009	0.00000031

Table 6.3 Backlight triangulation Gaussian curvature approximations.

The optimum seed triangle flattening results for each surface triangulation can be seen in Table 6.4. For comparison, Table 6.5 shows the flattening data for the regular RATC tests that were performed in Section 5.6.

Surface	ΔA (%)	Average ΔA (mm^2)	ΔS (%)	Average ΔS (mm)	Seed Success (%)
Backlight	0.452	0.378	0.131	0.548	88
Windscreen	0.16	0.196	0.052	0.299	90

Table 6.4 Principal curvature influenced triangulation flattening results.

Surface	ΔA (%)	Average ΔA (mm^2)	ΔS (%)	Average ΔS (mm)	Seed Success (%)
Backlight	0.475	0.4	0.137	0.591	92
Windscreen	0.178	0.209	0.058	0.322	90

Table 6.5 Regular RATC triangulation (regular grid) flattening results.

The flattened triangulations for the backlight and windscreen surfaces, for each AMPC influenced triangulation, can be seen in Figures 6.24. Each triangulation also has a colour shading, showing the triangles that had a change in absolute area. The optimum seed triangle positions are shown as solid black and white in the wireframe and colour shadings respectively. Figure 6.25 shows the flattening results generated from the regular RATC triangulations, using a uniformly sampled grid, from Section 5.6.

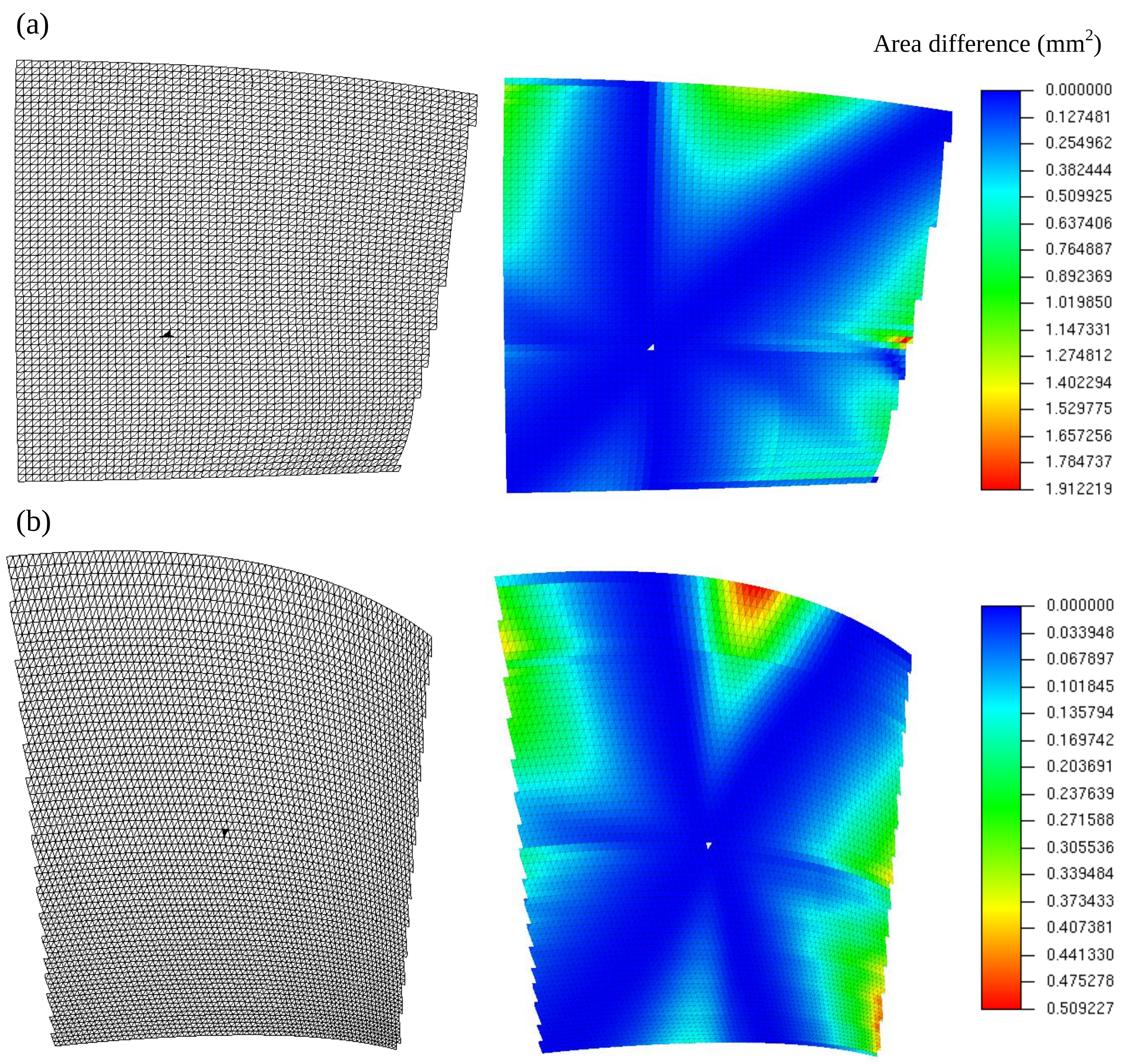
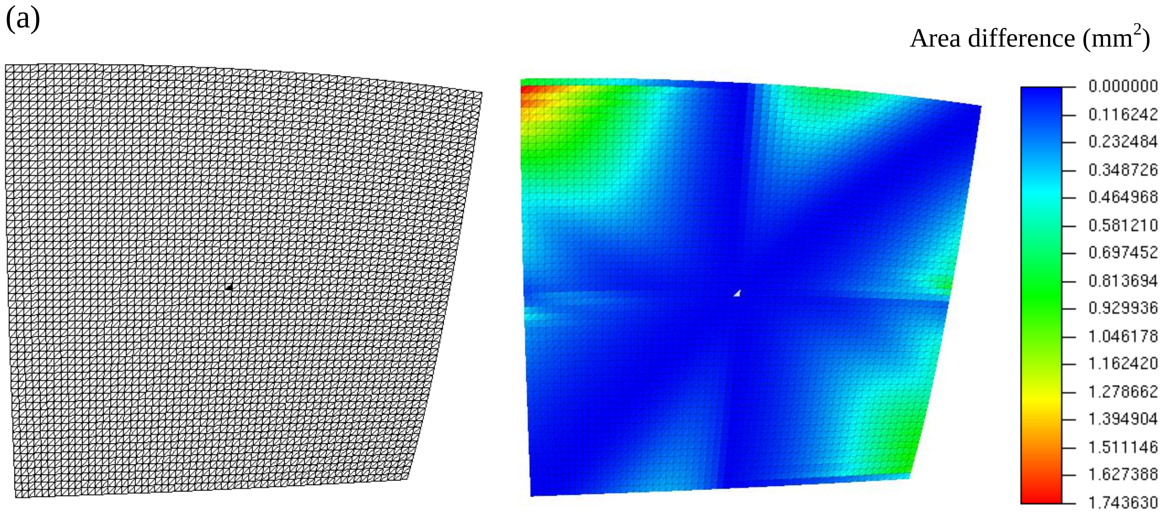


Figure 6.24 (a) Backlight and (b) Windscreen flattening results from AMPC influenced triangulations.

6.5.2 Gaussian Curvature Approximation Analysis

It can be seen from Table 6.3 that the AMPC influenced triangulations generated good approximations for the Gaussian curvature. For the backlight surface, K_{Gmin} was relatively underestimated by 5% and for the windscreen surface K_{Gmax} was relatively overestimated by 4%. These Gaussian curvatures cannot be directly compared to the regular RATC triangulation results from Chapter 5, as the curvatures are evaluated at different points

(a)



(b)

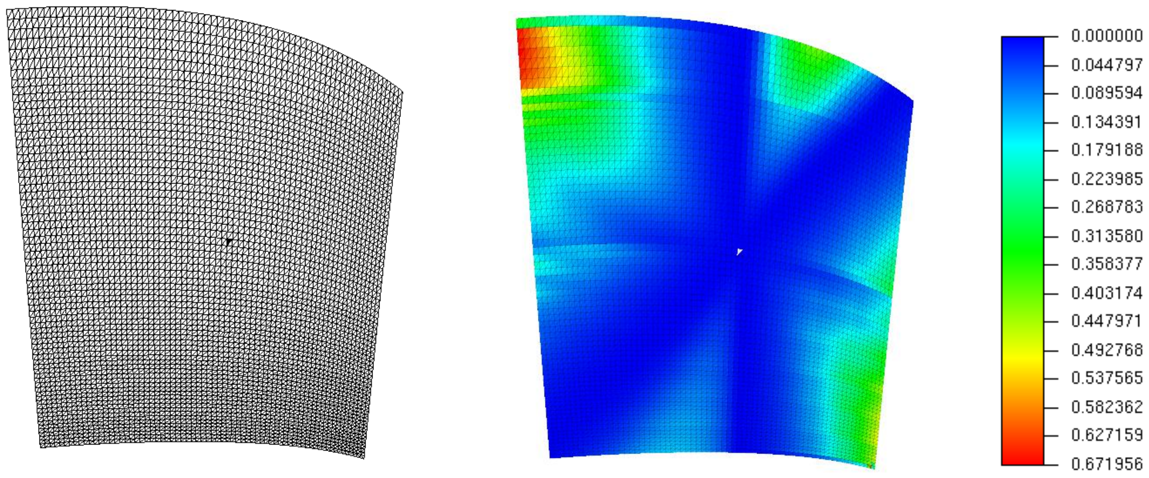


Figure 6.25 (a) Backlight and (b) Windscreen flattening results from regular RATC triangulations using a uniformly sampled grid of points.

along the surface, thus giving different Gaussian curvature values. However, it is noted that the average K_G difference per vertex, for the entire principal curvature triangulation, was less than the regular RATC triangulation, generated from a uniformly sampled grid, by 4% and 9% for backlight and windscreen respectively (Section 5.6.1, Tables 5.5 and 5.6). This is essentially indicating that the Gaussian curvature approximation across all points was closer to the actual values than the regular triangulation in Section 5.6.

The Gaussian curvature is the product of the minimum and maximum principal curvatures (Section 1.4.3). Therefore, by taking the influence of the MPCD when sampling points, the triangulation topological data inherited this characteristic more accurately, which resulted in a better approximation of the Gaussian curvature. This suggests that having the triangulation vertices influenced by both the directions of principal curvature could enhance the quality of the Gaussian curvature approximation. However, further research would need to be conducted for this to be established.

6.5.3 Flattening Analysis

In terms of the flattening results, the magnitude of area and shape distortion caused by the AMPC influenced triangulations was reduced, when compared with the regular RATC triangulations from Section 5.6. The amount of absolute area distortion has been reduced by 5% and 10% for the backlight and windscreen triangulations respectively. Furthermore, the magnitudes of the absolute shape distortion were also improved by 4% and 10% for the backlight and windscreen triangulations respectively.

These improvements are attributed to the fact that the AMPC influenced triangulations inherited more developability information by ensuring that one edge of every triangle followed along the modal AMPC. Therefore, when flattening these edges, they will undergo less geometrical change, as they follow along lines of minimum principal curvature. As the edges define the shape of the triangulation (Section 4.3), this will result in less shape distortion, which will cause less area distortion. This reduction in distortion can be seen in Table 6.4. Furthermore, the average area change per triangle and average shape change per edge was also improved from the regular RATC triangulation from Section 5.6.

Chapter 7

CONCLUSIONS AND FUTURE WORK

This thesis investigated triangulation and the downstream effects on surface flattening. Two novel topics were explored, notably RATCs and AMPC influenced triangulations, each of which was found to have an impact on the triangulation dependant method of surface flattening.

RATs are commonly used throughout research and literature (Section 2.10, Section 4.1). However, given a set of n cells, generated from uniformly sampled points, there are 2^n different ways in which the diagonals could be placed to form a final triangulation consisting of RATs. These different combinations of edges were introduced as RATCs. The choice of RATC is an ambiguity currently existing in literature. For example, McCartney *et al.* (1999) and Cadar *et al.* (2005) both present flattening algorithms using different RATCs. This can make it difficult to compare data as the RATCs are not consistent and the impact of each on flattening is not understood (Parwana and Cripps, 2009).

To investigate the effects of RATCs three global configurations were proposed, notably regular, diamond and chevron (Section 5.2). In addition, local variations of RATCs were also explored. Rather than randomly changing the local RATC of a triangulation, a new method was presented to locally fit RATCs that best approximated the local Gaussian curvature of the original surface (Section 5.4). The advantage of locally varying RATCs based on a surface characteristic was that a well defined solution would be generated, removing any ambiguities when choosing a RATC.

During the development of this optimised RATC algorithm, it was found that the current method used to approximate the Gaussian curvature from a triangulation (Section 5.5.2) did not work well for triangulations consisting of RATs with varying valencies (i.e. locally changing RATCs). Therefore, the Gaussian curvature approximation algorithm was improved by applying an area compensation factor (Section 5.5.5) to account for these valency changes.

The Gaussian curvature optimised RATC and three global RATCs were applied to two almost developable surfaces, which were then flattened and compared (Section 5.6). The flattening results presented, for each configuration, were those produced by the optimum seed triangle (i.e. seed triangle that induced the least absolute area change). For both surfaces, the optimised RATC triangulations generated the closet Gaussian curvature approximation to the original surface. The diamond configuration was found to generate the worst approximation because the vertex valencies of the points, within the triangulation, varied between 4 and 8, which are the extreme valencies (Section 5.6.2). From the three global RATCs, the regular configuration was found to produce the best Gaussian approximation, which was very close to the optimised RATC (Section 5.6.1).

In terms of flattening, the regular configuration was found to produce the least change in absolute area, followed by the optimised RATC. The diamond configuration was found to induce the largest absolute area distortion. The improved flattening results of the regular configuration were attributed to the fact that the valency of every triangle within the triangulation was the same. Unlike the diamond configuration, this consistent vertex valency ensured that a regular mapping order was generated when the triangulation was being flattened (Section 5.6.3). The power of the regularity of the vertex valencies was further highlighted by the fact that the regular configuration had the highest seed triangle success rate, making this RATC the most stable. The optimised RATC also had many varying valencies, which caused it, like the diamond configuration, to have a very low seed success rate. However, despite the low success rate, the flattening result generated by the optimised RATC was still good as it approximated the original surface characteristics well. Although this result was distinguished from the case studies in Section 5.6, further work would be suggested to perform an in-depth investigation into the influence of Gaussian curvature driven triangulations and the downstream effects on flattening. This study could provide valuable feedback, which could be used to create an improved method and error analysis for approximating Gaussian curvature from a triangulation.

In addition to seed success rates, the optimum seed triangle was commonly used throughout this thesis when comparing the performance of different triangulations on flattening. The seed triangle placement is an interesting factor and it was found that optimum seed triangle was generally located towards the centre of a surface. This is because it reduces the number of levels which are generated by the mapping order, reducing the number of triangles that need average, which reduces the amount of distortion. The method that was used to determine the optimum seed triangle was to trial every triangle within the triangulation as the seed, flattening the triangulation and then retrospectively choosing the triangle that induced the least absolute change in area. However, this method was very inefficient and time consuming for triangulations with large densities. Therefore, a suggestion for future work is to further investigate the optimum seed position and establish a method that identifies its position by analysing the shape of the triangulation.

The regular configuration was found to produce the best overall flattening result in terms of absolute area change and seed success rate. However, it was observed that the Gaussian curvature optimised RATC performed better in terms of absolute shape change. This was attributed to the fact that the Gaussian curvature optimised triangulation inherited the shape of the original surface better than the regular configuration. To explore this idea of surface shape preservation within a triangulation, another interesting study was performed on the influence of triangulation developability and the effects on flattening. Developable surfaces should flatten without inducing any area or shape distortion. However, it was shown that the transfer of this developabilty information from the surface to the triangulation can be lost (Section 6.1). It was established, through experimental evidence, that skewing the vertices of a triangulation in a particular manner, so that no triangle edges followed along the AMPC, would cause the flattening to induce distortion. As the shape of a triangulation is defined by its edges, by having no edges follow along the AMPC meant that the triangulation did not inherit this shape characteristic.

Therefore, a new method was proposed to triangulate a surface, whilst ensuring that one edge of every triangle followed the general direction of the AMPC (Section 6.3). This triangulation scheme was found to improve the surface flattening of an almost developable surface. By inheriting the developability of the original surface better, the absolute shape change of the triangulation after flattening was reduced when compared against the regular configuration triangulations of Section 5.6. It was also found that this triangulation approximated the Gaussian curvature of the original surface more accurately than this regular configuration (Section 6.2). It is suggested that ensuring that the edges follow both the axis of minimum and maximum principal curvature, may help improve the Gaussian curvature further. In addition, as the principal directions are always perpendicular at any given 3D point, this method may also be useful in generating triangulations that truly consist of RATs in real space. Therefore another suggestion for future work is to further investigate and understand the effects of preserving and maintaining the developability of a triangulation. The current method could be improved by using C^2 continuous polylines rather than just C^1 to improve the approximation for the AMPC. This work could also be extended by formulating a method that generates these polylines and classifies the developability before any flattening is performed. This investigation could be the key to establishing a definite measure to define an almost developable surface, which is still an ambiguity which remains within this area of work. For example, the curvature of the polylines generated from the AMPC could be a measure of the developability.

This thesis also introduced a new efficient and robust parametric trimmed surface triangulation method (Chapter 3). Efficiency was gained during trimmed curve tracing by minimising the number of cells processed. Key features were the efficient tracing algorithm and knowledge of orientation of the trimming curves is not required (Cripps and Parwana, 2011). The method is applicable to NUBS surfaces and operates on the untrimmed surface, constructing a rectangular parametric grid onto which the trimming curves are traced. This approach also minimises the occurrence of degenerate triangles and copes with holes independently of the grid size.

REFERENCES

de Berg, M., van Kreveld, M., Overmars, M. and Schwarzkopf, O. (2000), *Computational Geometry Algorithms and Applications*, Second Edition, Springer-Verlag, ISBN 3-540-65620-0.

de Boor, C. (1972), On calculating with B-Splines, *Journal of Approximation Theory*, 6, pp. 50-62.

Cader, R.A., Ball, A.A. and Cripps, R.J. (2005), Pattern shape design for ceramic ware: an integrated solution, *International Journal of Computer Integrated Manufacturing*, 19(3), 287-293.

Clarkson, K., Tarjan, R.E. and van Wyk, C.J. (1989), A fast Las Vegas algorithm for triangulating a simple polygon, *Discrete Computer Geometry*, **4**, pp. 423-432.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2001), *Introduction to Algorithms*, Second Edition, The Massachusetts Institute of Technology, ISBN 0-262-03293-7.

Cox, M.G. (1972), The numerical evaluation of B-Splines, *Institute of Mathematics and its Application*, 10, pp. 134-149.

Cripps, R.J. and Cook, P.R. (1999), Point based CADCAM, *Advances in Manufacturing Technology XIII*, *Proceedings of the 15th National Conference of Manufacturing Research*, University of Bath, 6-8 September, pp. 149-153.

Cripps, R.J. and Parwana, S.S. (2011), A robust efficient tracing scheme for triangulating trimmed parametric surfaces, *Computer-Aided Design*, 43, pp. 12-20.

Czerkawski, A.M. (1996), *Fitting Procedures for Curves and Surfaces*, Ph. D. Thesis, Geometric Modelling Group, The University of Birmingham, UK.

Delcam Plc (2010), www.delcam.co.uk, [26 June 2010].

Dyn. N., Hormann, K., Kim, S. and Levin, D. (2001), Optimizing 3D triangulations using discrete curvature analysis, *Mathematical Methods in CAGD: Oslo 2000 Nashville*, Vanderbilt University Press, pp. 135-146.

Faux, I.D. and Pratt, M.J. (1985), *Computational Geometry for Design and Manufacture*, Ellis Horwood Limited, ISN 0-8531-2114-1.

Field, D.A. (1991), A generic Delaunay triangulation algorithm for finite element meshes, *Advances in Engineering Software*, 13, pp. 263-272.

Foley, J.D, van Dam, A., Feiner, S.K. and Hughes, J.F. (1995), *Computer Graphics-Principles and Practice*, Second Edition, Pearson Education, ISBN 0-321-2105-65.

Frey, P.J, George, P-L. (2000), *Mesh Generation Application to Finite Elements*, HERMES Science Publishing, ISBN 1-903398-00-2.

Kumar, G.V.V.R., Srinivasan, P., Shastry, K.G. and Prakash, B.G. (2001), Geometry based triangulation of multiple trimmed NURBS surfaces, *Computer-Aided Design*, 33, pp. 439-454.

Kumar, S. and Manocha, D. (1995), Efficient rendering of trimmed NURBS surfaces, *Computer-Aided Design*, 27(7), pp. 509-521.

Kumar, S. and Manocha, D. (1996), Interactive Display of Large Scale NURBS models, *IEEE Transactions on Visualisation and Computer Graphics*, 2(4), pp. 323-336.

Lipschutz, M.M. (1969), *Schaum's Outline of Differential Geometry*, McGraw-Hill,
ISBN 0-070-37985-8.

Lockyer, P.S. (2007), *Controlling the Interpolation of NURBS Curves and Surfaces*, Ph. D. Thesis, Geometric Modelling Group, The University of Birmingham, UK.

Lohner, R. and Parikh, P. (1988), Generation of three-dimensional unstructured grids by the advancing front method, *International Journal for Numerical Methods in Fluids*, 8, pp. 1135-1149.

Luken, W.L. (1996), Tessellation of trimmed NURB surfaces, *Computer-Aided Geometric Design*, 13, pp. 163-177.

McCartney, J., Hinds, B.K. and Seow, B.L. (1999), The flattening of triangulated surfaces incorporating darts and gussets, *Computer-Aided Design*, 31, pp. 249-260.

Meek, D.S. and Walton, D.J. (2000), On surface normal and Gaussian curvature approximations given data sampled from a smooth surface, *Computer-Aided Geometric Design*, 14(6), pp. 521-543.

Nutbourne, A.W. and Martin, R.R. (1988), *Differential Geometry Applied to Curve and Surface Design (Volume 1: Foundations)*, Ellis Horwood Limited, ISBN 0-7458-0140-4.

O'Rourke, J. (2001), *Computational Geometry in C*, Second Edition, Cambridge University, ISBN 0-521-64976-5.

Parwana, S.S. and Cripps, R.J. (2009), Surface triangulation and the downstream effects on flattening, *Mathematics of Surfaces XIII,* 13th IMA International Conference, York, 7-9 September, pp. 293-306.

Peterson, J. W. (1994), *Tessellation of NURBS surfaces, In Graphics Gems IV*, Academic Press, ISBN 0-123-36155-9, pp. 286-320.

Piegl, L.A and Richard, A.M. (1995), Tessellating trimmed NURBS surfaces, *Computer-Aided Design*, 27(1), pp. 16-26.

Piegl, L.A. and Tiller, W. (1997), *The NURBS Book*, Second Edition, Springer-Verlag, ISBN 3-540-61545-8.

Piegl, L.A. and Tiller, W. (1998), Geometry based triangulation of trimmed NURBS surfaces, *Computer-Aided Design*, 30(1), pp. 11-18.

Pilkington Group Limited (2010), www.pilkington.com, [26 June 2010].

Riesenfeld, R.F. (1973), *Applications of B-Spline Approximation to Geometric Problems of Computer-Aided Design*, Ph. D. Thesis, Syracuse University.

Rockwood, A., Heaton, K. and Davis, T. (1989), Real-time rendering of trimmed surfaces, *ACM Computer Graphics*, 23(3), pp. 107-116.

Rogers, D.F. and Adams, J.A. (1990), *Mathematical Elements for Computer Graphics*, Second Edition, McGraw-Hill, ISBN 0-07-100289-8.

Sadoyan, H., Zakarian, A., Avagyan, V. and Mohanty, P. (2006), Robust uniform triangulation algorithm for computer aided design, *Computer-Aided Design*, 38, pp. 1134-1144.

Schneider, P.J. and Eberly, D.H. (2003), *Geometric Tools for Computer Graphics*, Morgan Kaufmann, ISBN 1-558-60594-0.

Schoenburg, I.J. (1946), Contributions to the problem of approximation of equidistant data by analytical functions, *Applied Mathematics*, 4(1), pp. 45-99; 112-141.

Shimada, K. and Gossard, D.C. (1998), Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis, *Computer-Aided Geometric Design*, 15, pp. 199-222.

Shu, C. and Boulanger, P. (2000), *Triangulating Trimmed NURBS Surfaces*, Curve and Surface Design: Saint Malo, Vanderbilt University Press, ISBN 0-826-51356-5.

Seidel, R. (1991), A simple and fast randomised algorithm for computing trapezoidal decompositions and triangulating polygons, *Computational Geometry Theory and Application*, **1**(1), pp. 51-64.

Sheng, X. and Hirsch, B.E. (1992), Triangulation of trimmed surfaces in parametric space, *Computer-Aided Design*, 24(8), pp. 437-444.

Shimada, K. and Gossard, D.C. (1998), Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis, *Computer-Aided Geometric Design*, 15, pp. 199-222.

Shu, C. and Boulanger, P. (2000), *Triangulating Trimmed NURBS Surfaces*, Curve and Surface Design: Saint Malo, Vanderbilt University Press, ISBN 0-826-51356-5.

Sugihara, K. and Inagaki, H. (1995), Why is the 3D Delaunay triangulation difficult to construct?, *Information Processing Letters*, **54**, pp. 275-280.

Versprille, K.J. (1975), *Computer-aided Deisgn Applications of the Rational B-Spline Approximation Form*, Ph. D. Thesis, Syracuse University, Syracuse, NY.

Wang, C.C.L., Smith, S.S-F. and Yuen, M.M.F. (2002), Surface flattening based on energy model, *Computer-Aided Design*, 34, pp. 823-833.

Wang, D., Hassan, O., Morgan, K. and Weatherill, N. (2006), EQSM: An efficient high quality surface grid generation method based on remeshing, *Computer Methods in Applied Mechanics and Engineering*, 195, pp. 5621-5633.

Weiler, K. and Atherton, P. (1977), Hidden surface removal using polygon area sorting, *ACM SIGGRAPH Computer Graphics*, 11(3), pp. 214-222.