Modelling and Animation using Partial Differential Equations

Geometric modelling and computer animation of virtual characters using elliptic partial differential equations

Michael ATHANASOPOULOS

Thesis submitted for the degree of

Doctor of Philosophy.

Centre of Visual Computing

School of Computing, Informatics and Media

University of Bradford

July 2011

TABLE OF CONTENTS

List of figures

List of tables

Acknowledgments

I would like to thank my numerous collaborators who helped make this dissertation possible. First, i thank my parents for their constant support in my studies.

My supervisors, Professor Hassan Ugail and Dr Gabriela González Castro deserve many thanks for their guidance and their encouragement to complete my studies.

I particularly thank both Dr Eyad Elyan, who initially developed the Maya plug-in presented in Chapter 4 for constructing PDE surfaces, and Hussein Alaskari who initiated the development of the Maya user interface for generating cyclic animation presented in Chapter 5.

Last but not least, I thank Claire Roberts for the love and understanding she has given me all throughout my studies.

Abstract

This work addresses various applications pertaining to the design, modelling and animation of parametric surfaces using elliptic Partial Differential Equations (PDE) which are produced via the PDE method. Compared with traditional surface generation techniques, the PDE method is an effective technique that can represent complex three-dimensional (3D) geometries in terms of a relatively small set of parameters. A PDE-based surface can be produced from a set of pre-configured curves that are used as the boundary conditions to solve a number of PDE. An important advantage of using this method is that most of the information required to define a surface is contained at its boundary. Thus, complex surfaces can be computed using only a small set of design parameters.

In order to exploit the advantages of this methodology various applications were developed that vary from the interactive design of aircraft configurations to the animation of facial expressions in a computer-human interaction system that utilizes an artificial intelligence (AI) bot for real time conversation. Additional applications of generating cyclic motions for PDE based human character integrated in a Computer-Aided Design (CAD) package as well as developing techniques to describe a given mesh geometry by a set of boundary conditions, required to evaluate the PDE method, are presented. Each methodology presents a novel approach for interacting with parametric surfaces obtained by the PDE method. This is due to the several advantages this surface generation technique has to offer. Additionally, each application developed in this thesis focuses on a specific target that delivers efficiently various operations in the design, modelling and animation of such surfaces.

Chapter 1: Introduction

1.1 Background

The growth of computer hardware and the progress of computational algorithms over the last few years have given us new alternatives for creating and manipulating 3-dimensional (3D) geometry. Complex and time consuming computations are now affordable and they are easy to produce through the current Computer-Aided Design (CAD) packages which provide various tools for the design and manipulation of complex geometries and surfaces in general. The latest advances in graphics hardware require new techniques to take advantage of the new hardware specifications and enable the programming of the rendering pipeline. Geometry can now be visualized and manipulated in real time in various mobile devices without requiring high performance specifications.

Some of the World Wide Web technologies such as Virtual Reality Modelling Language (VRML) provide capabilities for creating 3D models or virtual environments that can be embedded in web pages. Its applications vary from art to engineering while the main area of development is visualizing data. Additionally, new libraries such as the WebGL can process and display Hardware-assisted 3D rendering in web browsers without the use of plug-ins by utilizing the Graphics Processor Unit (GPU). Furthermore, there is a growing demand for new techniques that can utilize all the new possibilities that are introduced in the field of computer graphics. In the following chapters various techniques are presented that take advantage of a recently developed surface generation technique that produces a parametric surface as a solution to a suitably chosen Partial Differential Equation (PDE). This technique is known as

the PDE method and offer several advantages to different areas in computer graphics over other more traditional surface generation techniques.

1.2 Objectives

The major objectives of this work can be summarized as follows:

- To use Partial Differential Equations as a 3D surface generation technique.
- To utilize the PDE method in a CAD environment such as Autodesk Maya. This involves communication between various layers of a node based system such as Maya, using a scripting language and a high level programming language such as C++.
- To integrate of the PDE method in an interactive 3D environment for designing and manipulating Aircraft configurations.
- To explore various animation techniques utilizing the PDE method. Two techniques were developed for generating cyclic animation using PDE surfaces. The first technique focuses in creating everyday cyclic motions for a PDE based human model using simple mathematical expressions. Whereas the second one focuses in producing fish locomotion by means of manipulating the spine of the PDE surface.
- To develop an automatic curve extraction technique for converting mesh models to a set of boundary-based patches required in order to obtain a PDE-based surface representation for each patch comprising the original mesh model. This is a new approach of mesh segmentation that can potentially minimize the storage and memory requirements in a number of applications.

- To develop a human-computer interaction system that integrates facial expressions in the form of PDE-based surfaces.

1.3 Achievements

Each of these methodologies presents a new technique for interacting with parametric PDE surfaces. The main contribution of this thesis is to exploit the PDE characteristics over traditional polygonal and parametric surfaces. This is achieved by reducing the storage requirements of a given model to a small set of boundary based curves, providing tools for direct surface manipulation and deformation and offering build-in tools for the animation of the surface.

The adaptation of the PDE method in various platforms has simplified the construction and manipulation of such surfaces while creating the opportunity to communicate with several external tools, such as script and skeleton hierarchies, in order to achieve better results. Among the achievements, a new technique has been developed which aims to characterize a given mesh model into a set of curve based patches that are used to evaluate the PDE method. The advantage of this technique is that a given mesh model can be represented as a set of patch wise PDE surfaces, thus utilizing all the PDE characteristics. This methodology can be proved very useful for introducing such surfaces in LOD systems where the resolution of a model is dependant in various criteria such as camera distance and topological various characteristics.

The PDE method has also been proved very useful in the animation of surfaces. To that extend, several techniques have been developed that exploit the characteristics of parametric PDE surfaces in generating cyclic motions and blend shapes. Utilizing the PDE spine, a by-product of the analytic solution of the PDE method, provides the user with a tool for creating and manipulating

parameterized dynamic animation. The animation can be controlled using simple periodic functions whereas different motion characteristics can be modelled intuitively in a CAD environment.

Moreover, a computer-human interaction technique has been developed to adapt PDE surfaces in order to minimize the data transfer and re-use the animation of face expressions to various given face models. The contribution of the thesis can be summarised as the development of several new techniques with a shared objective, to utilize the characteristics of parametric PDE surfaces. The following chapters will present each technique in details followed with several examples.

1.4 Publications

Below is the list of journals and conference publications that were published.

- M. Athanasopoulos , H. Ugail and G. González Castro (2009): "Parametric design of aircraft geometry using partial differential equations" *Advances in Engineering Software* , **40** 479-486.
- M. Athanasopoulos, G. González Castro and H. Ugail (2009): "Cyclic Animation of Human Body Using PDE Surfaces and Maya", *2009 International Conference on Cyberworlds, IEEE Computer Society. ISBN: 978-0-7695-3791-7* , University of Bradford, UK.
- G. González Castro, M. Athanasopoulos M and H. Ugail (2010): "Cyclic Animation using Partial Differential Equations" *The Visual Computer*, **26** (5): p. 325-338.

- M. Athanasopoulos, G. González Castro and H. Ugail (2010): “On the development of an interactive talking head system” *International Conference on Cyberworlds 2010.* p.414-420
- M. Athanasopoulos, G. González Castro and H. Ugail (2010):”*On the development of a talking head system based on the use of PDE-based parametric surfaces”. Transactions in Computational Science XII,ISBN: 978-3-642-22335-8.*

More detailed the above publications can be found in the chapters below:

- Chapter 3: Parametric design of complex surfaces using PDE
    - Automatic curve extraction
- Chapter 4: Manipulation of PDE surfaces
    - A toolkit to design and manipulate aircraft geometry
    - Global Manipulation using the PDE Maya plug-in
- Chapter 5: Generating PDE-based animation
    - Cyclic animation of human motion
    - Fish Locomotion using the PDE spine
- Chapter 6: Interactive talking head system
    - Building a PDE based talking head system

1.5 Resources

A wide selection of applications and data where used for developing the work were presented in this thesis. The development of the various tools which will be discussed in the next chapters, was undertaken with the use of C++ and C# in Visual studio 2005 as well as OpenGL for visualization of the PDE surfaces; whereas the modelling, manipulation and animation of several models throughout this thesis was accomplished with the use of a CAD package such

as Autodesk Maya. Additionally, open source libraries were used for developing the interactive talking head system for converting text to speech using the Microsoft SAPI 5 and the Rebecca AIML API for generating a response from a given input text. All the models were purchased or constructed manually with the use of Maya.

1.6 Structure of thesis

The thesis is structured as follows:

- Chapter 2, "**Introduction to geometric modelling**", aims to give an introductory overview of implicit and parametric surfaces and discuss their advantages and disadvantages. The second part of this chapter gives an overview of the PDE method including the mathematical theory underneath it and it is illustrated by various graphical examples. This methodology will be used as the primary surface generation technique throughout this thesis.
- Chapter 3, "**Parametric design of complex surfaces using PDE**". This chapter presents three curve fitting techniques that have been used in this thesis to describe a given mesh surface as a set of parametric curves that will be evaluated by the PDE method. Moreover, this chapter focuses primarily on a new curve extraction technique which its purpose is to automatically segment a given surface to a set of curves that satisfy the boundary requirements of the PDE method.
- Chapter 4, "**Manipulation of PDE surfaces**" presents two different applications that integrate the PDE method in interactive 3D environments. The first application is responsible for the design and manipulation of different aircraft configurations using parametric PDE

surfaces. The second application has been developed as an external plug-in for the Autodesk Maya package and it continues the work of a previously developed plug-in used for constructing PDE surfaces within the Maya environment. Its purpose is to expand the existing functionality and introduce global and local deformations on a given surface by constructing an underlying invisible PDE surface layer that surrounds and interactively deforms the surface in question.

- Chapter 5," **Generating PDE-based animation**" examines various animation techniques used in computer graphics. It covers some basic theory behind commonly used techniques in animating traditional polygon surfaces. The second part of this chapter focuses on exploiting the PDE method in animation and particularly on producing cyclic motions for two different examples. The first example consists of producing cyclic motion for basic movements such as walk and run of a PDE based human character model using the Maya environment. The second example utilizes the PDE spine as the control skeleton for simulating fish locomotion of PDE-based models.
- Chapter 6,"**Interactive talking head system**" presents a technique for generating PDE based facial expressions used in a computer-human interaction environment. The proposed methodology utilizes various tools to produce an environment that will interact with a user to produce realistic conversations. The system consists of four layers responsible for producing the computer-user interaction; the artificial intelligence (AI) engine, the text-to-speech (TTS) engine, the blend shape layer and the PDE solver for producing the final animated surfaces.

Chapter 2: Introduction to geometric modelling

2.1 Introduction

Given that the demand for more detailed 3D models in virtual environments is growing, various techniques have been developed to generate such surfaces. The importance of these techniques is that they can approximate a set of control points to create a smooth parametric curve; the approximated curves will be evaluated later by various surface generation methodologies such as the PDE method. This chapter will be covering the theory behind several polynomial curve and surface generation techniques.

2.2 Types of curves

2.2.1 Polynomial curves

A 2-dimensional curve is defined parametrically as,

$$Q(u) = (X(u), Y(u)), \tag{2.1}$$

where $X(u)$ and $Y(u)$ are the x and y coordinates of a point on the curve in question for any value of u. Figure 2.1 shows an example of a polynomial curve. Polynomials are computational efficient and easy to work with, however it is not always possible to produce a satisfactory curve using single polynomial for $X(u)$ and $Y(u)$ [1]. To that extend, a higher degree is required in order to satisfy a large number of constraints. For example, a polynomial of degree $(n-1)$ is needed to fit a polynomial curve though n data points. However, higher degree polynomials are not efficient to process and are numerically unstable [2].

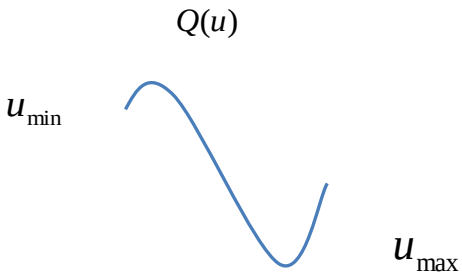


Figure 2.1. A parametrically defined curve.

One of the solutions to this problem consists of making used of the fact that a curve can be broken into a number of pieces called segments. Each segment is defined by a separate polynomial where it connects its segments together to form a polynomial curve, Figure 2.2. A polynomial curve can be defined based on an interpolation or approximation technique. In the case of interpolation the curve is required to pass through the data points P_i. However in the case of an approximation based technique, the curve uses the data points as guidelines so that the best fit approximation is found and therefore, it does not necessary pass through the data points.

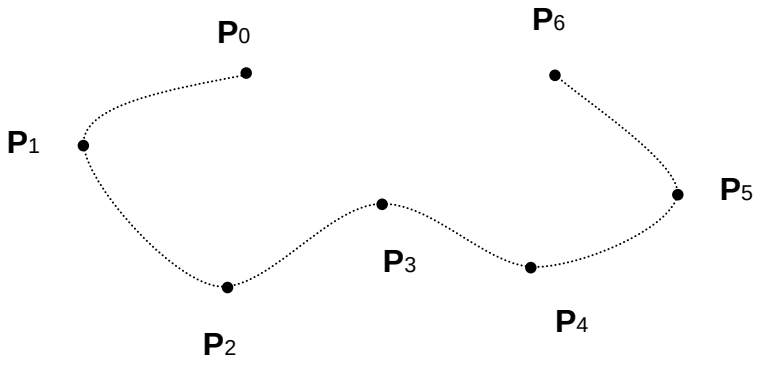


Figure 2.2. An example of an interpolated polynomial curve passing through all the control points.

The use of polynomial interpolation results in some undesired features. The computation of the interpolating polynomial for higher degree is computational

expensive compared to linear interpolation. Additionally, artifacts can appear at the end points of the curve when using a higher degree polynomial interpolation, a problem called Runge's phenomenon [3]. To that extend, splines-based techniques, such as Bézier or B-splines are commonly used to overcome these limitations.

2.2.2 Splines

Spline is a special function defined piecewise by polynomials [1]. A piecewise function is a function whose definition changes depending on the value of the independent variable. Splines are popular curves used widely in many CAD packages [2] for designing and modelling of parametric surfaces. They offer simplicity in their construction, accuracy of evaluation, and can approximate complex shapes through various curve fitting techniques. Spline interpolation is preferred to polynomial interpolation because it returns the same results even when using low-degree polynomials. Cubic splines usually generate good approximations while their evaluation is computational efficient. Additionally, a Spline can be categorized according to the polynomial used for evaluation, for example a B-spline is evaluated using the basis function on the entire curve; a Bézier spline is evaluated using Bernstein polynomials; and Hermite spline is using the Hermite polynomial for interpolating each segment.

2.2.3 B-splines

B-spline or Basis spline is a curve parameterised by a spline function that is evaluated using the De Boor algorithm [4]. They consist entirely of smooth

curves; however, sharp corners can be generated by joining two spline curve segments. Although, a B-spline curve consists of a set of control points, it generally does not pass through them. B-spline is a generalisation of a Bézier curve and it can avoid the Runge phenomenon without increasing the degree of the polynomial. A particular property of a B-spline is the local control point; it allows altering only a small part of the curve by re positioning a single curve point V_n . The local control points or control points are connected by straight lines to form the control polygon as seen in Figure 2.3; and are usually applied in situations of approximation rather than those related to interpolation. Figure 2.3 shows the control points, $V_0 \ldots V_n$, required to evaluate the cubic spline curve.

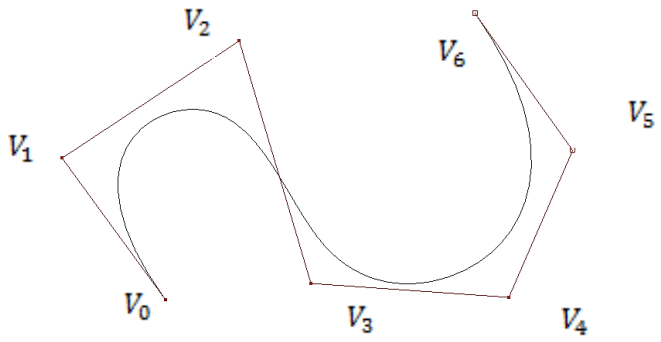


Figure 2.3. Example of cubic b-spline curve

The addition of control points enables the possibility of a curve partially without affecting the remaining segments; a very useful property for the design and modelling of parametric curves and surfaces.

A p th degree B-spline is defined by [4],

$$C(u) = \sum_{i=0}^{n} f_i(u) P_i, \tag{2.2}$$

where P_i are the control points, and $f_i(u)$ are the n^{th} degree piecewise polynomial functions. Although B-spline curves require more information, and a more complex mathematical formulation than Bézier curves; they offer more advantages compared to Bézier curves. A B-spline is a special case of Bézier curve. A B-spline can be a Bézier curve and satisfy all important properties that Bezier curves have. The degree of a B-spline is not dependant on the number of control points, thus a lower degree curve can still contain a large number of control points. They can provide more control flexibility by manipulating the position of a control point locally without changing the overall shape of the curve. However, polynomial curves cannot represent simple shapes such as ellipses or curves; thus a more generalized B-spline is required.

2.2.4 Bézier splines

Another type of spline curve is the Bézier curve [4, 5]; it is a parametric curve, first developed in 1962 by the French engineer Pierre Bézier. Although originally they were used by Renault to design automobile bodies, currently they are widely used in Computer graphics to model smooth surfaces. A linear, 1st degree Bézier curve is usually a straight line between two given points V_0 and V_1. Higher degree curves requires more intermediate control points between the two endpoints to match position, gradient and curvature. In order to mach only position and gradient requirements, a 2nd degree polynomial is sufficient. Example of a higher degree curve is the quadratics, 3rd degree Bézier curve, which is defined by three given points V_0, V_1 and V_2. The first two control points V_0 and V_2 are the two end points of the curve whereas the last control point V_1 is an intermediate point that controls the direction of tangents of both endpoints. The Quadratic and cubic degree Bézier curves are the most commonly used in

CAD packages for modelling and animation. Given that evaluating higher degrees Bézier curves are computationally expensive multiple low degree Bézier curves are used as a solution. These curves are then patched together in order to produce complex shapes. Each subdivided segment can be represented by a lower degree Bézier curve. A curve that is made of several Bézier curves is called a composite Bézier curve or a Bézier spline curve.

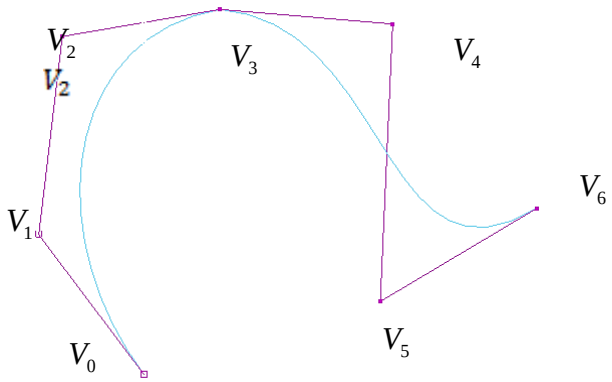


Figure 2.4.Example of cubic Bezier curve

Bezier curves are defined using four control points, also known as knots. Knots V_0 - V_3 are responsible for approximating a curve that passes through these control points. Control points V_1 and V_2 control the overall shapes of the curve, whereas V_0 and V_3 are the endpoints of the curve.

Figure 2.4 shows a Bézier spline curve containing a two patched Bézier curves. There are different methods for forming such curves. The Bézier-Bernstein method gives an approximation to a mathematical function that passes through a given set of control points. This is more suitable for interactive curve design since the control points give the designer a better control over the curve shape.

A degree d Bezier is define in [4] as,

$$Q(u) = \sum_{i=0}^{d} B_{i,n}(u) P_i, \qquad 0 \le u \le 1 \tag{2.3}$$

where the Basis function $\{B_{i,n}(u)\}$ are the n^{th} degree Bernstein polynomial, and P_i are the coefficients or control point. Bezier curves have various properties for controlling the curve shape. Some of them are:

- The curve passes through the start and end points of the control polygon.
- The curve follows the shape of the control point polygon and lies within the convex hull of the control points. In the case of a closed curve, the first and last control points are the end points of the curve segment.
- The order of the curve is related to the number of control points. Hence using many control points to control the curve shape results in evaluating high order polynomials.
- The curve is transformed by applying affine transformations to its control points and generating the new curve from the transformed control points.

2.3 Types of surfaces

Geometric surfaces consist of polygonal, implicit and parametric. As with the curve generations techniques, surface generation techniques can be defined with an implicit or parametric equation. Each of these surfaces has their advantages and disadvantages and will be outlined in this section.

2.3.1 Polygonal surfaces

A polygonal surface, which is the most common type of surface used in computer graphics, consists of a collection of vertices, faces and edges connected to each other to form a polygonal mesh. Edges are defined by two vertices that are connected to each other by a straight line, whereas three vertices connected to each other by three edges form a triangular face. Since faces are planar, they cannot bend; however smooth surfaces can be approximated by many small flat surfaces, this is usually achieved by subdividing the original face in smaller faces. Although polygonal surfaces are faster to visualize than any other representation, they are incapable of manipulating the surface directly. Figure 2.5, contains an example of a mesh object; it is impossible to directly manipulate a complex surface like the polygon mesh below without applying various complex and slow computational deformation techniques.

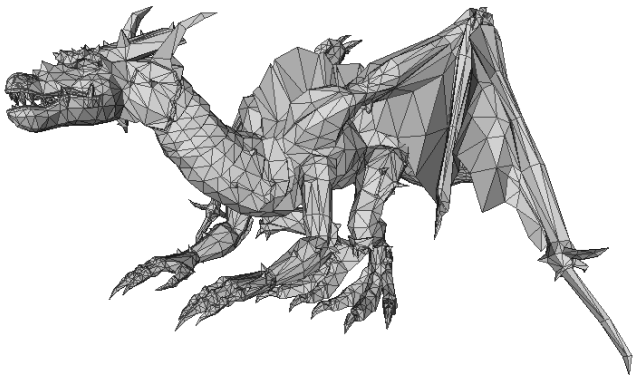


Figure 2.5.Example of a polygonal mesh model of a dragon [6].

2.3.2. Implicit surfaces

Implicit surfaces have been used in various fields such as mathematics, physics, biology and engineering. They are defined mathematically as a set of points x that are satisfying the implicit function $f(x)=0$; where x is a point on the surface implicitly described by the function $f(x)$ [7]. In other words, point x is on the surface if and only if the relationship $f(x)=0$ holds for x. This representation is called implicit since it provides a test for determining whether a point is on the surface or not, however it does not give any explicit rules for generating such points. In 3D space an implicit surface is given by a function [8],

$$f : R^3 \longrightarrow R, \tag{2.4}$$

such that, for an iso-value v, the implicit surface, is defined where

$$f(x,y,z)=v$$

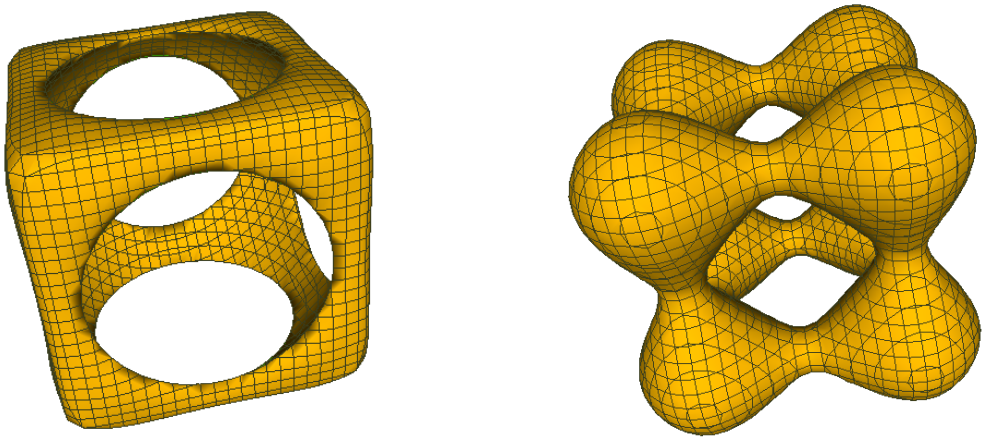
The implicit representation of a surface is constructed with properties that if:

$f(x,y,z)=0,$ point is on the surface.

$f(x, y, z) < 0,$ point is inside.

$f(x, y, z) > 0,$ point is outside. (2.5)

Implicit surfaces can represent geometry in various ways. For example it can represent a surface as a set of zero functions which can be specified by discrete samples or blobby functions, mathematical functions or procedural methods. Discrete samples can be used to measure physical properties of geometry such as colour, density, temperature and pressure. Whereas mathematical functions can include any mathematical expressions for generating a surface. The most known algebraic surfaces are the quadratic ones; these surfaces offer efficient rendering with only a few intrinsic parameters for controlling its shape.



(a) (b)

Figure 2.6. (a) Example of implicit cube-sphere surface. (b) Blob surface.

The surface represented in Figure 2.6 (a) is mathematically defined using the Equation 2.6.

$$\left(((\frac{1}{2.3})^2(x^2+y^2+z^2))^{-6}+((\frac{1}{2})^8(x^8+y^8+z^8))^{6-1}\right)=0, \quad (2.6)$$

whereas as Figure 2.6 (b) has been computed using the Equation 2.7.

$$\left(x^2+y^2+z^2+\cos(4x)+\cos(4y)+\cos(4z)-0.2\right)=0, \quad (2.7)$$

where x, y and z are coordinates of the Cartesian system and $f(x,y,z)=0$.Both surfaces were computed using the K3DSurf surface generation application [9].

There are various techniques allowing the rendering of implicit surfaces, some of them include polygonization, ray-tracing and contours.

Polygonization [10] occurs by spatial partitioning of the surface into smaller voxels and using marching cubes algorithm to identify the polygons needed to represent the part of the iso-surface that passes through this cube. The new individual polygons are then merged with the resulting surface. The technique can be summarized as follows, given an iso-surfaces defined by $f\{x\}=0$ where $f(x,y,z)=v$.

- Subdivide 3-dimensional space into uniform cells/voxels
- Evaluate implicit function for all grid points
- Find surface intersection voxels
- Subdivide surface-intersecting voxels to threshold size
- Polygonize the voxel based on the edge intersections.

One disadvantage of using this technique is that the quality of the polygonized mesh is dependant to the cell size. For example, if the cell size is too large, the surface might not be able to be defined properly. Topological errors might

appear and features might not be contained in the output polygonized surface. Alternatively, if the cells are too small the technique is not efficient, computational expensive and prone to numerical errors.

Ray-tracing techniques [11] usually visualize the surface by subdividing the space in which the surface lies in into progressively smaller voxels and using ray tracing techniques to identify and visualize the voxels that are intersecting the surface. Finally, contour techniques [12] are used to reconstruct an iso-surface from cross sectional contours. Each cross section is a non-intersecting polygonal shape consisting in one or several closed polygonal contours.

Implicit surfaces offer many advantages over the construction, manipulation and visualization of a surface; some of them are:

- They offer efficient check whether a point is inside or outside a surface, evaluating for $f(x, y, z)$ as seen in equation 2.5, can identify if a point is inside, outside or on the surface.

- Surface normals can be computed at any given point by the partial derivatives as,

$$n(x, y, z) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}), \tag{2.8}$$

  where $\frac{\partial f}{\partial x}$ is the partial derivative of a function f with respect to variable x and n is the normal vector .

- Implicit representations offer efficient surface intersections. Given a ray with a starting point p and a direction v, all points on the ray can be expressed in

parametric form as $x(t) = p + tu$. Inserting this ray equation into the implicit representation gives $f[x(t)] = 0$ which it can be solved for t.

- Additionally, implicit surfaces can offer efficient Boolean operations. The arithmetic operators from equation 2.5 are replaced with Set operators to allow direct manipulation of an iso-surface. Examples can be seen using Constructive Solid Geometry (CSG) [13]: using this technique, an iso-surface is represented as a tree with simple implicit functions as its leaf nodes and boolean operators as its interior nodes. The geometric primitives that form the Boolean operations are soft objects bounded by the iso-surface.

2.3.3. Parametric surfaces

Parametric surfaces are polynomial based methods that can be used to generate complex geometries by using a set of control points that controls the shape of the surface. The parametric representation, can be written as ,

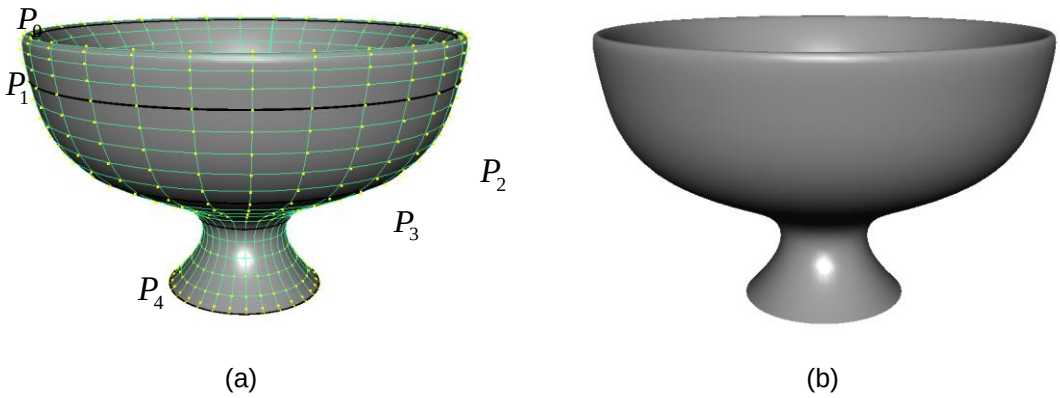
$$X = F(u, v), \tag{2.9}$$

where u and v are surface parameters, X is a vector x, y, z denoting a point on the surface and $0 \le u \le 1$. Compared to implicit surfaces, parametric representations allow direct generation of points on the surface by setting the values of u and v where $X(u, v) = F(u, v)$.

Parametric surfaces are ideal for generating polygonal surfaces that approximate a given set of boundary points. Figure 2.7 (a) contains an example of a parametric surface computed using the NURBS function [5] in Maya, for the

given boundary curves P_0.....P_4, as seen in Figure 2.7 (b).The boundary conditions can be defined for x, y and z by the parametric function such a,

$$\begin{aligned} x &= f_x(u,v) \\ y &= f_y(u,v) \\ z &= f_z(u,v), \end{aligned} \tag{2.10}$$



(a) (b)

Figure 2.7. Boundary conditions and control net (a).
Example of a parametric NURBS surface (b).
Surface manually designed in [9].

The normal vector $\vec{n}$ for a parametric surface can be computed from the cross product of any two linearly independent vectors that are tangent to the surface at that point, Equation 2.11.

In order to calculate $\vec{n}$ we need the partial derivatives of x, y, z with respect to u, v . If $X(x,y,z)$ is a vector representing the point x, y, z then $X_u(u,v) = (x_u, y_u, z_u)$ and $X_v(u,v) = (x_v, y_v, z_v)$ are vectors of the partial derivatives of that point.

Then Normal $\vec{n}$ is defined by,

$$\vec{n} = \left(\frac{\partial_x}{\partial_u} \times \frac{\partial_x}{\partial_v} \right), \tag{2.11}$$

since the surface is explicitly parameterized, every point on the surface contains parametric coordinates that can be used to generate texture coordinates for indexing a texture map. Figure 2.8 contains another example of parametric surface used to define a simple terrain representation. Compared to polygonal meshes, design and manipulating complex parametric surfaces becomes an easy task; adjusting only the control points of the boundary conditions results in smooth surface representations.

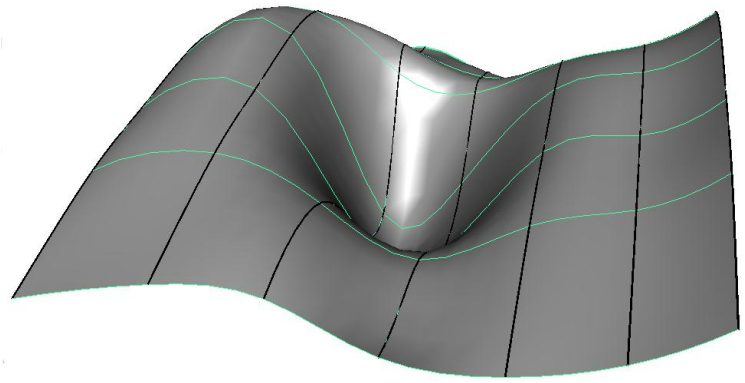


Figure 2.8. Example of a NURBS surface used to define a terrain surface. Surface manually designed in [9].

However, a parametric surface is difficult to ray-trace, since there is no direct way to take an arbitrary point in space and test it to see if it is on the surface. Another disadvantage of parametric surfaces is that they require piecewise parametric representation to represent complex shapes. This is usually done by partitioning the surface into parametric patches; the same technique used for constructing parametric spline curves. Each parametric patch is defined by a set of control points used to describe the surface and a parametric function is used to calculate the new surface points. Three popular techniques used for generation of parametric surfaces are presented below.

2.3.3.1 Bezier surfaces

As with the Bézier curve, the Bernstein basis function is used for the Bézier surface patch. A Bezier surface is the tensor product of Bezier curves [4]. To create a Bézier surface, we blend a mesh of Bézier curves using the blending function,

$$S^{w}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u) B_{j,m}(v) P_{i,j}^{w}, \qquad (2.12)$$

where *j* and *k* are points in parametric space and $P_{x,y}$ represents the location of the knots in real space. The $B_{i,n}(u)P_i$ and $B_{j,m}(v)P_j$ are the Bezier functions that extend to rational Bezier surfaces for u and v. The Bézier functions specify the weighting of a particular sequence of knots know as the Bernstein or Bezier coefficients.

A Bezier surface patch can be generated by a set of $(n+1)(m+1)$ control points. Similar to the case of the Bezier curve, every point on the surface depends on u and v parameters which vary between 0 and 1. The parameter u controls the variation from top to bottom along the patch ($P_{0,j}$ - $P_{n,j}$), whereas parameter v controls the variation from left to right ($P_{i,0}$ - $P_{i,m}$) as specified in Equation 2.12. Calculating the normal for a point on the surface is a simple operation; it is computed by taking the derivative of the surface with respect to u and v separately, which represents the tangent vectors of the surface in the direction of either u or v. The normalized cross product of these tangent vectors will result in the surface normal for a particular point. The u and v tangent vectors can be calculated separately using the Equation 2.13.

The normal vector N_X can be calculated using the cross product of T_u and T_v,

$$N_X = T_u \times T_v, \tag{2.13}$$

where the tangent of u is calculated by $T_u = \frac{\partial X_{(u,v)}}{\partial_u}$ and the tangent of v is computed by $T_v = \frac{\partial X_{(u,v)}}{\partial_v}$.

2.3.3.2 B-Spline surfaces

The B-spline surface is defined by a set of points or control net, two knot vectors and the product of the B-spline functions [4, 14],

$$S\ (u,v) = \sum_{i=0}^{n}\ \sum_{j=0}^{m} B_{i,p}(u) B_{j,q}(v) P_{i,j}, \tag{2.14}$$

where $B_{i,p}(u)P_i$ and $B_{j,q}(v)P_j$ are the p_{th} and q_{th} degree B-spline functions defined by the knot vectors $(n+1)$ and $(m+1)$. A B-spline surface is another example of tensor product surfaces where the set of control points are usually referred to the control net and the u and v parameters range between 0 and 1.

B-spline surfaces contains several important properties, some of them are:

- They contain a strong convex hull property inherited from B-spline curve. This guarantees that the resulting surface patch will lie completely in the convex hull of the control points $P_{i,j}$.
- The local modification property of B-spline curves exists for the surfaces as well. If a control point $P_{i,j}$ is transformed, only the neighbouring area on the surface of the moved control point will be affected from the change.

- Affine Invariance. Applying affine transformations directly to the control points that describe a surface will transform the surface as well.
- A B-spline surface can become easily a Bezier surface.

2.3.3.3 NURBS

NURBS (Non Uniform Rational Basis Splines) surfaces, are another example used for generating parametric surface patches [5]; they are derived from extending the NURBS curve equation to two parametric directions u and v. A NURBS surface of p[th] degree in u direction and q[th] degree in the v direction it is a piecewise rational function of the form,

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)w_{i,j}}, \qquad 0\le u,v\le 1 \qquad (2.15)$$

where $P_{i,j}$ is the control net, $w_{i,j}$ are the weights and the $N_{i,p}(u)$ and $N_{j,q}(v)$ are the non-rational Basis spline functions. By evaluating this equation we obtain a surface with local control and one knot vector in each parametric direction and with rational properties. Rational functions allow better control over the derivatives of curves, which results in adjusting the curvature of the surface [4]. Figure 2.9 contains an example of a NURBS surface; the curves P_0, P_1 and P_2 are the boundary conditions required to evaluate the rational B-spline functions.

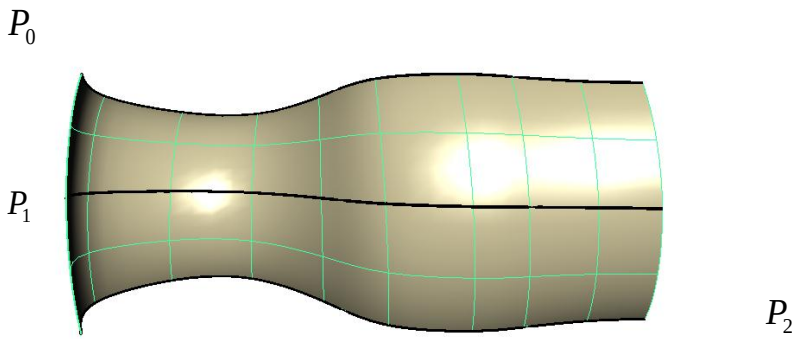


Figure 2.9.Example of a NURBS surface and its boundary conditions, curves P_0, P_1 and P_2.

The use of parametric and implicit functions for representing a surface has both their advantages and disadvantages. To achieve successful geometric modelling is usually done by utilizing both techniques. Some of the advantages and disadvantages of the two methods are listed below.

- Parametric representation is more suitable for designing and representing complex shapes in a 3D environment; the coefficients of a given parametric form (e.g. Bezier, B-splines, NURBS) provides many advantages over the design and manipulation of a surface.
- The complexity of many geometric operations is dependant to the selection of the representation form. For example, using the parametric representation is easy to enumerate points on the surface but requires more complex operations to identify if a point is on the surface or not. From the other hand, utilizing the implicit form offers easier surface intersection test whereas it is more difficult to generate a new point on the surface.

2.4 The use of PDE as a surface generation technique

2.4.1 Introduction

PDE have been recently introduced in computer graphics as a tool for geometric modelling and animation. A partial differential equation is an equation in which the unknown function depends on set partial derivatives of this unknown function with respect to two or more independent variables. It is an efficient boundary based surface generation technique that can represent complex 3D geometries in terms of a relatively small set of design variables. PDE can generate both implicit and parametric geometry. Examples of implicit PDE surfaces can be seen in free-form surface construction, noise reduction [15] and image manipulation [16] implementations. Whereas, parametric PDEs offer a lot of advantages in areas such as surface generation, shape blending [17], interactive design [18] and interactive sculpting [19].

The use of this methodology allows a surface to be regarded as graphical representation of the solution of a given partial differential equation subject to a particular set of boundary conditions, which serves as constraints.

The Bloor-Wilson PDE method is a method for generating parametric PDE surfaces and has been used throughout this thesis. The Bloor-Wilson PDE method [20], which was originally introduced as a blending tool [17], is a surface generation technique that offers many advantages compared with other polynomial surface generation techniques such as B-splines or NURBS. Some of them are:

- PDE-based surfaces require a smaller number of parameters compared to spline-based techniques for representing a given surface. PDE-based

surfaces can be characterized by a set of boundary curves, whereas spline-based techniques are evaluated using a set of control points.

- Surface smoothness can be guaranteed when blending two or more surface patches. The smoothness of a blended PDE-based surface can be increased from the order that is used for computing the PDE method.
- PDE-based surfaces can unify the geometric and physical aspects of surface modelling and manipulation.

2.4.2 Introduction to PDE method

The boundary value approach characterizing the PDE method is summarized mathematically below. However, the full mathematical details are presented in [21] and will give the reader a full overview if there is further interest in such detail. Let $\underline{X}(u,v)$ be a function defining a surface in 3D space in a domain Ω (with specified boundary data).

Here u and v represent the parametric coordinates of a point in Ω, and $\underline{X}(u,v)$ as a mapping from that point in (u,v) to a point in 3D space such that $R^2(\Omega) \rightarrow E^3$. We regard $\underline{X}$ as the solution to a partial differential equation of the form,

$$\underline{X}(u, v) = (x(u, v), y(u,v), z(u,v)), \tag{2.16}$$

thus, the full problem consists of finding a solution of the form,

$$\left(\frac{\partial^2}{\partial u^2} + a^2 \frac{\partial^2}{\partial v^2} \right)^r \underline{X}(u, v) = 0, \tag{2.17}$$

where a is an intrinsic parameter and r determines the order of the PDE.

Note that when $a=1$ and $r=2$ equation (2.17) is known as the biharmonic equation that can be used to model some physical phenomena occurring in areas such as fluid and solid mechanics. A wide variety of methods exist for finding the solution of elliptic PDE similar to that shown in Equation (2.17). These include elementary separation of variables, Green's Functions, and sophisticated numerical techniques, [22, 23].The approach undertaken in this thesis is based on the approximate solution to Equation (2.17). This solution is given in terms of analytic functions.

As seen in [21], taking the region Ω of (u, v) parameter space to be the region corresponding to ($0 \le u \le 1, 0 \le v \le 2\pi$), Equation (2.17) is solved as subject to the boundary conditions on the solution which relate how $\underline{X}(u, v)$ and its normal derivatives $\partial X/\partial n$ vary along $\partial\Omega$. The imposed boundary conditions on the solution are of the form,

$$\underline{X}(0,v) = \underline{P}_0(v), \tag{2.18}$$

$$\underline{X}(1,v) = \underline{P}_1(v), \tag{2.19}$$

$$\underline{X}_u(0,v) = \underline{\partial}_0(v), \tag{2.20}$$

$$\underline{X}_u(1,v) = \underline{\partial}_0(v), \tag{2.21}$$

The boundary conditions $\underline{P}_0(v)$ and $\underline{P}_1(v)$ determine the value of $\underline{X}(u,v)$ on the surface patch at $u=0$ and $u=1$, respectively. Whereas the derivative conditions $\underline{\partial}_0(v)$ and $\underline{\partial}_1(v)$ are determined from the surface normals at the corresponding boundaries of the surface at $u=0$ and $u=1$. Since the derivative vector is defined using a curve defined in the 3-dimensional space, the shape and position of this curve relative to the positional boundary curve determine both the direction and the magnitude of the derivative vector. Figure 2.10 below

contains an example of a PDE surface patch evaluating four boundary curves using a 40 x 40 resolution in the u, v parametric space.

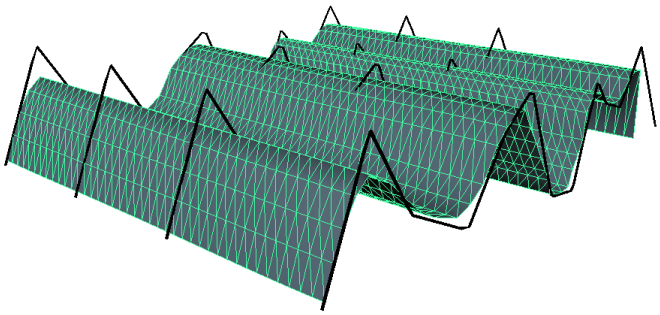


Figure 2.10. PDE surface resulting from a set of boundary conditions comprising piecewise linear polynomials. The resulting PDE surface was computed using a 40 x 40 grid.

In order to generate a typical PDE surface, Equation (2.17) is solved over a finite region Ω of the (u, v) parameter space subject to the boundary conditions with periodic boundary conditions, v being the periodic parameter, and using the method of separation of variables. Thus, the analytic solution of Equation (2.17) can be written as,

$$\underline{X}(u, v) = \underline{A}_0(u) + \sum_{n=1}^{\infty} \left[\underline{A}_n(u) \cos(nv) + \underline{B}_n(u) \sin(nv)\right], \qquad (2.22)$$

where

$$\underline{A}_0 = \underline{\alpha}_{00} + \underline{\alpha}_{01}u + \underline{\alpha}_{02}u^2 + \underline{\alpha}_{03}u^3, \qquad (2.23)$$

$$\underline{A}_n = \underline{\alpha}_{n1}e^{anu} + \underline{\alpha}_{n2}ue^{anu} + \underline{\alpha}_{n3}e^{-anu} + \underline{\alpha}_{n4}ue^{-anu}, \qquad (2.24)$$

$$\underline{B}_n = \underline{b}_{n1}e^{anu} + \underline{b}_{n2}ue^{anu} + \underline{b}_{n3}e^{-anu} + \underline{b}_{n4}ue^{-anu}, \qquad (2.25)$$

and $\underline{a}_{n1}, \underline{a}_{n2}, \underline{a}_{n3}, \underline{a}_{n4}, b_{n1}, b_{n2}, a_{n3}$ and b_{n4}, are vector-valued constants, whose values are determined by the imposing boundary conditions at $u = 0$ and $u = 1$. For a given set of boundary conditions, it is necessary to express the boundary conditions in terms of their respective Fourier series expansion, and identify the

various Fourier coefficients in order to find a one to one correspondence between the Fourier coefficients of the boundary conditions and the constants a_i and b_i respectively. For an approximate solution, the Fourier series can be truncated (typically $N = 6$) and a remainder term is included. Thus, Equation (2.22) leads to,

$$\underline{X}(u, v) = \underline{A}_0(u) + \sum_{n=1}^{N}\left[\underline{A}_n(u)\cos(nv) + \underline{B}_n(u)\sin(nv)\right] + \underline{R}(u, v), \tag{2.26}$$

Although the solution is approximate, the method guarantees that the chosen boundary conditions will be exactly satisfied by using the remainder function $\underline{R}(u, v)$ where is defined as,

$$\underline{R}(u, v) = \underline{r}_1(v)e^{wu} + \underline{r}_2(v)ue^{wu} + \underline{r}_3(v)e^{-wu} + \underline{r}_4(v)ue^{-wu}, \tag{2.27}$$

and $w = N(a+1)$ and $\underline{r}_1, \underline{r}_2, \underline{r}_3, \underline{r}_4$ are responsible for satisfying the original boundary conditions that are obtained by considering the difference between the original boundary conditions and the boundary conditions satisfied by the function

$$\underline{F}(u, v) = \underline{A}_0(u) + \sum_{n=1}^{N}\left[\underline{A}_n(u)\cos(nv) + \underline{B}_n(u)\sin(nv)\right] \tag{2.28}$$

Note that the Fourier series associated with the solution of Equation (2.17) is considered as an approximate solution satisfying the given boundary conditions. However, it can be considered as an exact solution if all the boundary conditions can be expressed in terms of a finite Fourier series. Figure 2.11 shows a sequence of figures that illustrate the effect of derivative condition $\underline{X}_u$ on the shape of the surface. Note that all the surfaces shown in Figure 2.11 have the same boundary conditions on the function $\underline{X}$ at $u = 0$ and $u = 1$,

whereas the boundary conditions on the function $\underline{X_u}$ at $u=0$ and $u=1$has been varied.

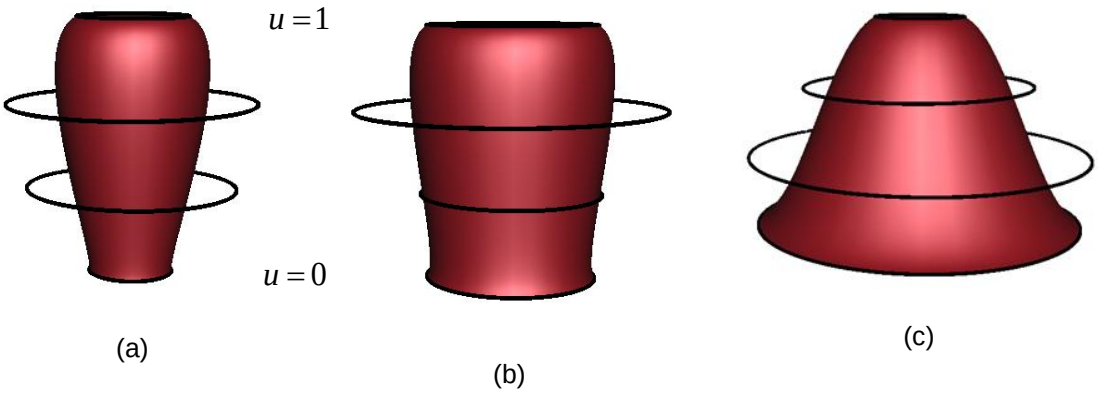


Figure 2.11. The effects of derivative conditions on a PDE surface (resolution 40 x 40 in the u , v parametric space).

Once the PDE method has been computed, a surface mesh can be generated as a solution of a suitably posed boundary value problem implemented on a 2D parametric space, a meshing scheme can be integrated with the aid of the 2-dimensional parameter space to obtain a suitable surface triangulation [24]. Another example of a PDE surface is presented below in Figure 2.12 (a) containing a closed surface. The last boundary curve P_3 has been scaled down and it is responsible for closing the surface. Figure 2.12 (b) shows a PDE-based representation of a closed cylinder consisting of three 4th order PDE patches; the base, top and body of the cylinder.

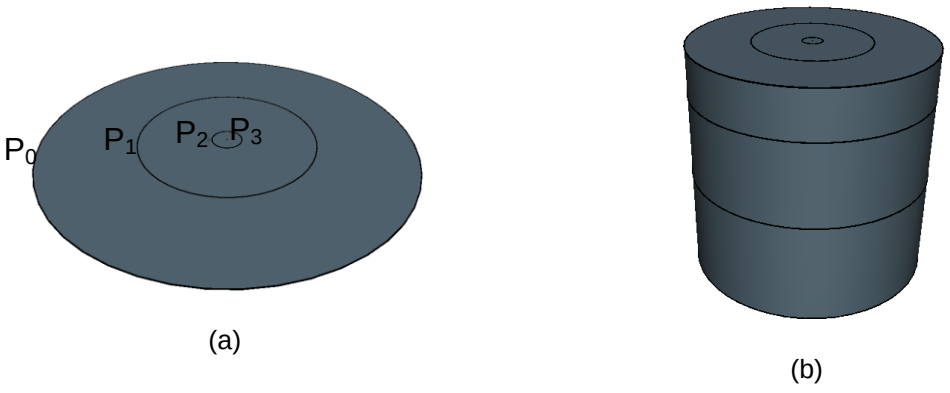


Figure 2.12. PDE-based representation of closed cylinder. Boundary curves associated with the top of the cylindrical surface (a) and (b) shows the full cylindrical surface

2.4.3 Applications of PDE surfaces

PDE can represent several physical phenomena. It can be used to represent physical models where the acceleration and velocity of the surface can be included such as the surface can be deformed according to external forces. Alternatively, it can describe the heat distribution in one or two dimensions in a given space. To that extend, PDE have been used under different approaches that vary from the computation and manipulation of a surface to addressing problems such applying local and global deformations using sculpting and blending of multiple PDE surface patches [25].

2.4.3.1 Interactive manipulation of PDE surfaces

Once a 3D surface has been defined using the PDE method, it may be necessary to manipulate it in order to improve or adjust the shape. The PDE method formulated by Bloor and Wilson [20] has been utilized as the foundation for the development of such surface generation tools. The derivative conditions determine the overall shape of the surface, whilst the size and the direction of the derivative vector can be adjusted interactively in order to manipulate the shape of the surface. Figure 2.11 shows the effect of derivative condition X_u on the shape of the surface. An important aspect for interactively manipulating a PDE surface is that the user does not required to have any mathematical knowledge concerning the PDE, since it is only required to define the boundary conditions that will represent the outer contour of the surface. The manipulation of the boundary conditions can be handled using simple geometric transformations such as translate, scale and rotate in an interactive environment [26].

2.4.3.2 Blending PDE surface patches

Another important function of the PDE method is surface blending of multiple patches [21]. In order to create more complex shapes we need to be able to create models that will contain more than one surface patch. Once a surface patch has been created, the PDE method can seek the solution of the next surface patch by evaluating the set of curves that belongs to the adjacent surface patch. The first step consists of identifying the exact position on a surface where the portion needs to be blended, as illustrated in Figure 2.13 (a). Any chosen point in the (u,v) parametric space will have an associated point on the surface. Thus, by creating a new curve on the parametric domain (u,v), the points on the surface are mapped onto the points in the parameter space. Note that any curve in the parametric space is guaranteed to lie on the surface. The new curve can be then manipulated interactively by the user. The next step requires adding additional boundary curves in order to define the new PDE patch adjacent to that surface. Figure 2.13 (b) shows the new boundary curves that are used to evaluate the new blended PDE patch as seen in Figure 2.13 (d). Note that this process must be carried out in such a fashion that a certain degree of smoothness is guaranteed. To that extend, the PDE method offers a solution for addressing this problem, where the degree of smoothness can be controlled by the order of the PDE used to find the surface. Additionally, a smooth blending between two surface patches can be guaranteed by either having common boundary conditions at the joining region of each path or a boundary curve lying on the surface patch as seen in Figure 2.13 (d).

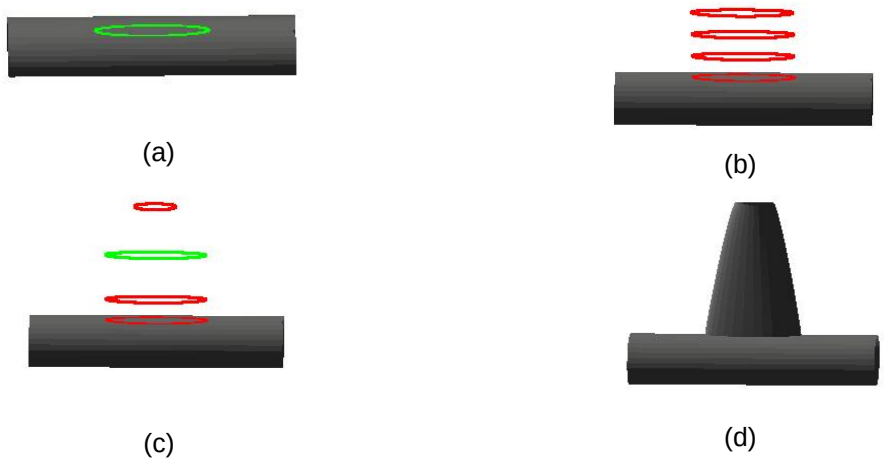


Figure 2.13. Blending PDE surfaces. Curve projected in the parametric space. (a) Creating the area that will be blended. (b) Adding a curve-set to evaluate the blended PDE-based surface (c) Manipulate a curve to achieve required design. (c). Surface blended with the new PDE surface patch

2.4.3.3 Trimming PDE surfaces

Another functionality of the PDE method is surface trimming [21, 27]. This follows a similar principle to that applied to surface blending is adapted, an area for removal needs to be identified by constructing a curve that lies on the parametric domain. Once the curve has been constructed, a bounding box containing the curve in the parameter space is computed. All the mesh points in the u,v space that belongs to this bounding rectangle are then indentified and discarded. A separate u,v mesh is then calculated in the new region between the rectangle and the curve in the parameter space, allowing a re-parameterization of the new surface. A linear interpolation, between points on the rectangle and the corresponding points on the curve, is carried out to determine the new u,v points for the re-parameterization. Finally, the new corresponding surface points and surface normals for the new parameterisation can be calculated.

2.4.4 Other applications

PDE have been used in various areas such as surface processing of a portion or an entire surface to areas such as surface manipulation and animation. Some examples of surface processing consist of image inpainting, noise reduction and N-sided hole filling are listed below.

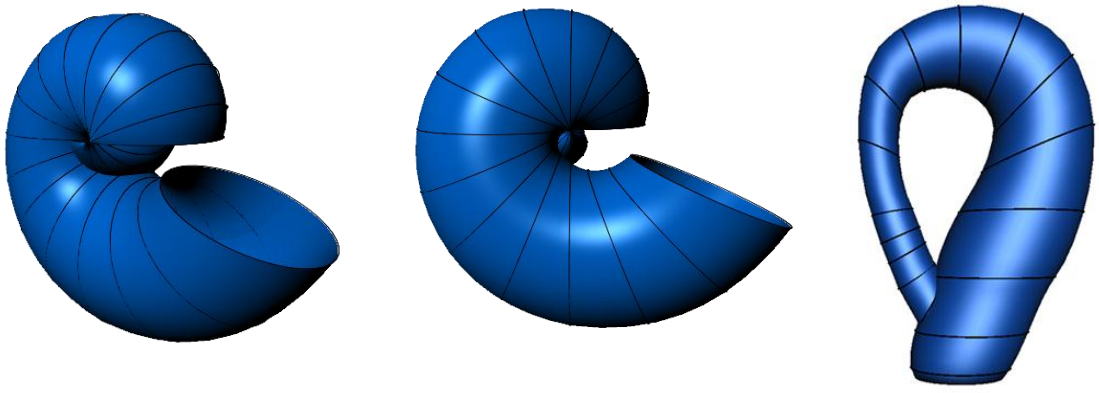
- Image inpainting is the technique used for modifying an image by removing an unwanted object. Implicit PDE have been used in [16] for such purpose.
- Parabolic PDEs in [15] have been used for reducing noise while preserving the image details.
- N-sided hole filling consists of constructing a surface for filling surface holes while guarantying surface continuity at the boundary. An example of such a technique can be seen in [28] where mean curvature flow and fourth and sixth order flows have been used to construct implicit PDE surfaces for filling such holes.

Another area where PDE surfaces can offer a lot of advantages is animation. Techniques such as morphing of facial expressions and generating cyclic animations using mathematical expressions or by utilizing the spine of the surface [29] are some of the examples that will be discussed in later chapters of this thesis. Parametric PDE surfaces appear to be very useful for implementing such techniques since the surface is determined by a set of boundary curves. This technique, capable of manipulating only a small set of control points can produce a quick and smooth transition between two objects. Additionally, it offers a natural mechanism for animating a surface by taking advantage of a mathematical property related to the solution of Equation 2.17. The spine of a PDE-based surface is analytically defined and corresponds to the term A_0 in

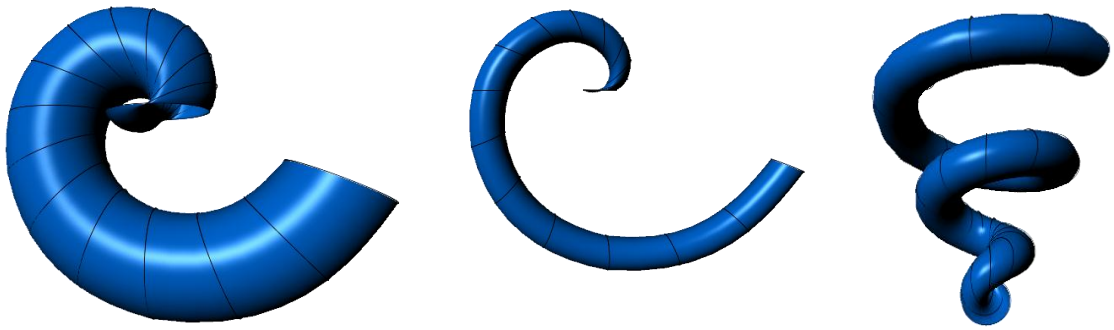
Equation 2.22. The spine can be thought as the skeleton or backbone of the geometry of a surface that is controlled through parameters defined on a simple analytic function. More information about cyclic animation using the PDE spine will be given in Chapter 5.

2.5 Examples of PDE surfaces

Figure 2.14 contains several screenshots of various surface representations computed using the PDE method. The surfaces have been computed in a 40 x 40 grid in the parametric domain. For each PDE surface contained in the figure below, the black curves are the boundary conditions required to evaluate the PDE method. The resulting surfaces are multiple blended surface patches, each of which is the graphical solution to a 4^{th} order PDE.



(a) (b) (c)



(d) (e) (f)

Figure 2.14. Examples of complex PDE-based surfaces. Representing a collection of shells (a, b, d, e, and f). Representation of the Klein bottle (c).

Figure 2.15 contains an example of a face model computed using 28 boundary curves as seen in Figure 2.15 (b). The resulting surface in Figure 2.15 (a) consists of 9 PDE surface patches blended together to ensure surface continuity. This example is used in a later chapter to generate facial expressions in a computer-user interactive talking head system. Figure 2.15 (c) and (d) contains the torso of the human body that is used in a later chapter to generate human based cyclic animation of a character.

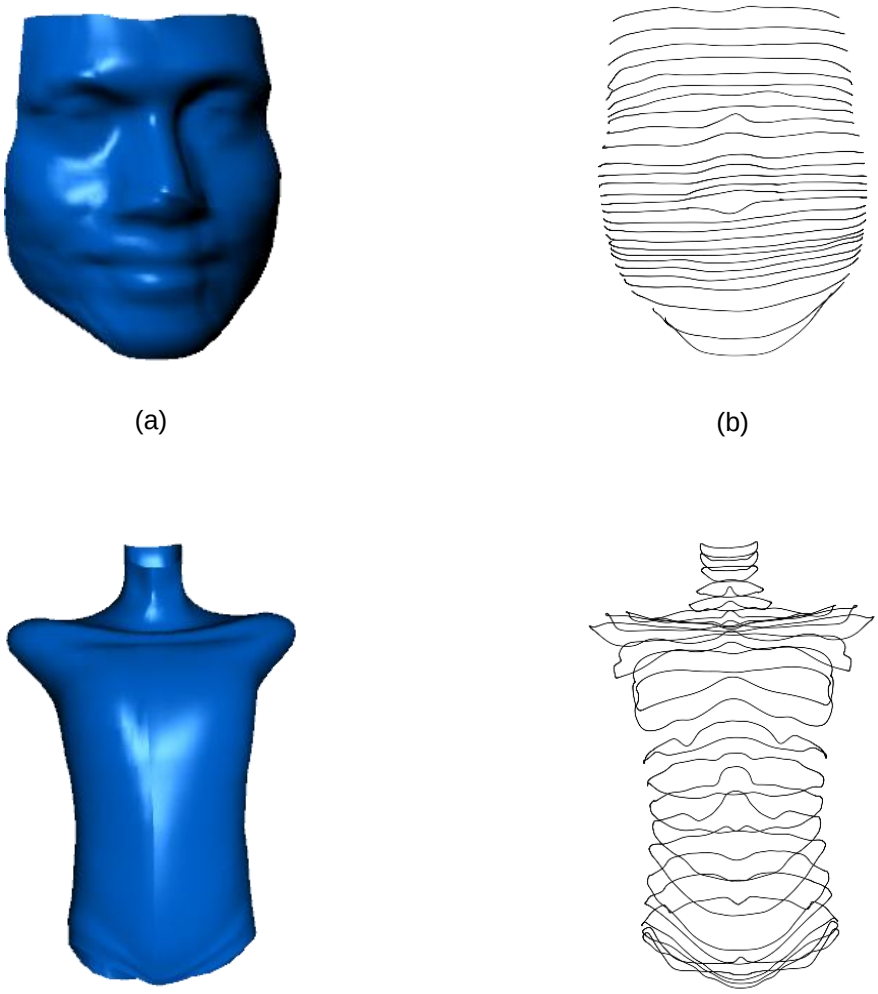


Figure 2.15. Example of a face model computed using the PDE method.
PDE-based surface representation (a, c).
Boundary conditions for computing the PDE method (b, d).

Figure 2.16 below contains various aircraft configurations designed using the PDE method in an interactive environment that implements a technique capable of direct design and manipulation of PDE-based aircraft shapes. Each configuration is split into three main groups, the fuselage, the latter wings and the rear wings; each group can be designed and manipulated real time by the user. Further information about this work can be found in Chapter 3 of this thesis.

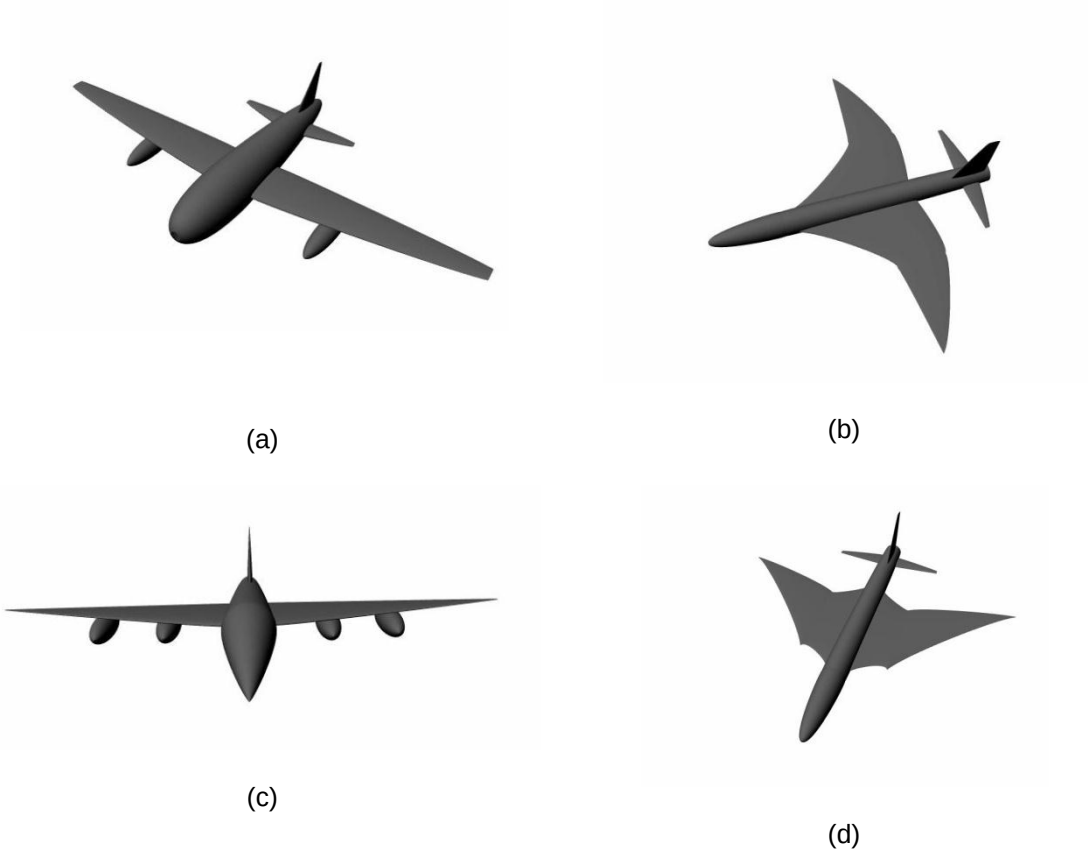


Figure 2.16. Examples of various aircraft configurations computed using the PDE method.

2.6 Conclusions

This chapter has presented a surface approximation technique used in computer graphics for generating parametric surfaces. The surface generation is based on the use of the PDE method and is fully parameterized to allow direct construction and manipulation of PDE based surfaces. Unlike other surface generation techniques, the PDE method can parameterize complex surfaces in terms of a small set of design variables, instead of hundreds of control points. Taking a boundary value approach to the problem of surface design has the advantage that most of the information required to define a surface are contained at the boundary conditions or contour curves representing that surface.

The PDE method offers several advantages over the design and manipulation of a surface. It can guarantee smooth blending between two surface patches by either adjusting the order of the PDE solution or by associating the boundary conditions with the parametric space between the two surface patches that need to be blended. Additionally, this chapter presents a new approach for using parametric PDE surfaces in an interactive environment for designing and manipulating PDE-based aircraft configurations. This technique provides the user with tools for simplifying the design of aircraft configurations using parametric PDE surfaces in a stand-alone 3D environment. Each new aircraft design can be saved and loaded again from the user interface for further manipulation. Whereas, the manipulation of each configuration is achieved by simple affine transformations that are applied directly to the boundary conditions of the PDE method.

Future work can be undertaken in improving the overall quality of the aircraft configurations. This can be achieved by including a variety of aircraft parts and transformation tools for improving the construction process of the airplane designs. Future work could also be undertaken in determining the impact of external static or dynamic forces to the materials and structures during the flight. In the following chapters, the PDE method is utilized in several applications focusing on modelling, manipulation and animation of PDE surfaces.

Chapter 3: Parametric design of complex surfaces using PDE

3.1 Introduction

A key step for producing PDE surfaces is obtaining the boundary conditions or curves that are required to compute the geometry. These curves can be defined either manually or extracted automatically by using various curve fitting techniques. In both cases, the quality of the output surface will be associated with the accuracy with which the curves represent the original surface. A smooth surface can be approximated using various approaches; most of the 3D surfaces in this work have been produced using a combination of cubic Bézier curves [5]. Bézier curves are parametric polynomial curves that are widely used in computer graphics to generate smooth curves; they consist of a set of control points or control handles that are used to directly to manipulate the curve by applying various affine transformations. Every control point in the curve is computed as a weighted sum of all the control points, this way each point is influenced by every control point in the set.

3.2 Curve extraction techniques

As it was mentioned in the previous section, choosing a higher degree polynomial can guarantee better approximation of position, gradient and curvature of the curve. However, there are cases where a 1^{st} degree polynomial curve can approximate better results by linearly interpolating several control points as curve segments. In this section we will focus on techniques used to generate several types of polynomial curves that are used to calculate the boundary curves required to obtain PDE surfaces.

3.2.1 Manual design of boundary curves

In this section a human 3D body will be used as a guideline to generate a set of curves. These curves will describe the human body surface and will be later used to calculate a set of PDE surfaces that will represent the original geometry [30]. Extracting the curves manually is a technique where the designer will be required to generate manually a set of curves representing a given object within a CAD package. An example is shown in Figure 3.1 were the character model was imported in the Maya (CAD package) as a guideline. The goal here is to identify meaningful regions and divide the mesh accordingly. Each of these regions will be represented with a PDE surface and used in the next chapter to generate cyclic animation.

Figure 3. 1.Original human geometry used for curve extraction.

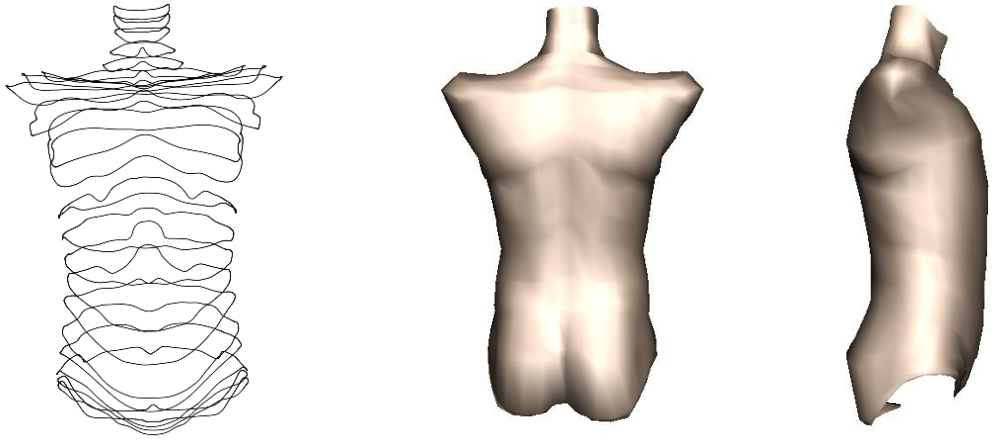
3.2.1.1 Polygon edges to curve technique

The human mesh model is divided in 5 regions; torso, left-right arm and left-right leg. The process consists of selecting a set of polygon edges connected to each other and converting them to a set of linear polynomial curves. Linearly interpolated curves are used to preserve the original curvature of the geometry

model representing the human body, where each selected edge becomes a curve segment of the overall curve. Thus, for every two points contained in a selected edge will be used as the control points for the final curve. After extracting enough curves to represent the geometry, the curves must be re-computed to hold the same number of control points and grouped according to the body part they are associated with. Figure 3.2, shows the left arm mesh region with the corresponding boundary curves; the arm consists of 16 Bezier curves that will be used to generate 5 PDE surface patches blended together.

Figure 3. 2.Boundary curves extracted from the left arm of the mesh model.

One of Maya's capabilities is it provides tools for manipulation and over surfaces and curves. Various tools can help the user visually create, select and modify any loaded data set from within the environment. Additionally, Maya comes with an integrated scripting language, MEL script, allowing even more control over any shape that exists in the system. The curve design process requires the extraction and manipulation of curves from the given surface geometry. The selection of polygon edges is used as a guideline to extract the outline shape of the area and convert it in to a set of curves. Once a set of edges is selected, with the use of various MEL scripts it can be converted to a curve and stored as a Maya NURBS object for later use. When enough curves are extracted, they are grouped and stored for the surface generation procedure.



(a) (b) (c)

Figure 3. 3.Torso region curve set and PDE surface (side and back view).

Figure 3.3 (a), contains the boundary curves representing the torso area of the human mesh model. Figure 3.3 (b) and (c) contains the side and back view of the PDE surface obtained from these boundary curves. A MEL script handles the procedure through which the curve data are exported or imported between Maya and an external C program required for calculating the PDE surface.

Figure 3.4, shows the boundary curves and the resulting PDE surface divided in 5 regions. The surface below is a close approximation of the original human body mesh model. In order to achieve a better representation, various curve fitting techniques must be applied and a PDE of higher order needs to be computed. Representing any geometry using parametric curves gives us the advantage of reducing the original data size since only a small set of control points is used to represent a surface instead of a large number of vertices and faces. Additionally, the surface can be re-calculated for a given LOD (Level of Detail) by updating the u,v space. This surface generation application can therefore produce meshes with different resolutions of the same object depending on the distance between the camera and the user or similar parameters.

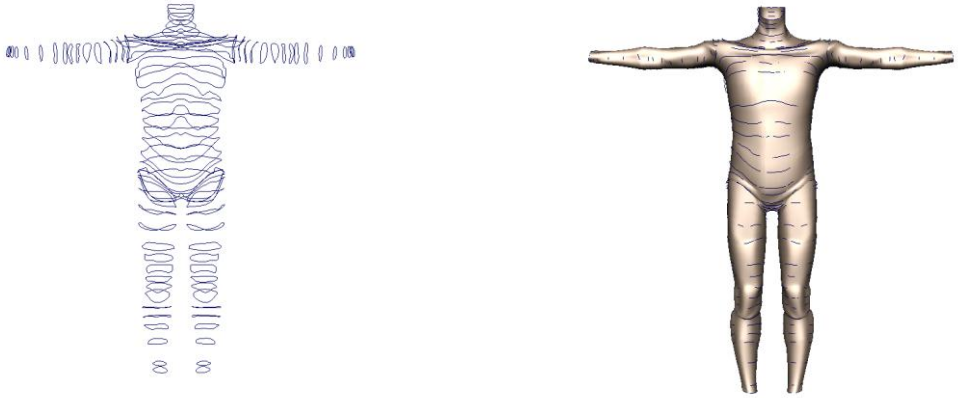


Figure 3. 4. The extracted boundary conditions of the human body (a) and the resulting PDE-based surface representation (b).

3.2.2 Parametric design of boundary curves

In this section a different approach is examined. This approach allows the user to design and manipulate curves used for constructing PDE-based aircraft geometry [31]. The process consists of generating a set of parametric curves that represents a particular aircraft shape. The aircraft is divided into 3 different basic parts; each part is a separate process that is required for the final design of the aircraft. The first step consists of generating the fuselage part of the aircraft. The process starts with an initial curve-set containing 4 curves that represent a cylinder, as a base for the fuselage part. Each curve can be manipulated interactively by applying simple transformations such as translation, rotation and scaling. The system can load existing pre-configured curves, where various properties such as distance, radius and position can be controlled by the user. The rest of the airplane is divided into wings and tail wings; each part is designed and manipulated using the same process. This technique is examined in more detail as a toolkit to design and manipulate aircraft geometry in Chapter 4 Section 2.

3.3 PDE-based mesh segmentation

3.3.1 Introduction

This section focuses on a new technique which aims in representing a given mesh model to a set of boundary-based curves that can be used to compute a set of parametric patch wise PDE surfaces. This is a unique methodology that can be proven very effective in solving the problem of converting a model to a parametric surface. Once a surface can be represented parametrically, it opens the possibility for utilizing the PDE characteristics for the manipulation and animation of such surface. In order to understand the problem of automatically generating boundary-based that will represent any given mesh model, first we need to be able to analyze and decompose the surface in a geometrical fashion. This problem lies in understanding how mesh and shape segmentation [32, 33] works. In the case of mesh segmentation, the model is segmented into a number of patches that are uniformly divided according to some property, such as curvature, geodesic distance etc. However, in the case of shape segmentation, the mesh model is divided into patches that identify parts that correspond to main features of the given shape, e.g. legs, arms, head, and torso for a human body mesh model. There are many approaches available that apply both techniques; however the solution is usually based on solving an application specific problem. The choice of the right segmentation algorithm is dependent on the application requirements. For example, mesh based approaches usually require that the boundaries of the regions must be smooth; and the boundaries where the regions meet should allow continuity with the neighbouring regions. Additionally, the mesh simplification algorithm that is proposed in this work is required to divide the shape into uniform curve-sets that will be used to calculate a parametric surface instead of mesh patches.

3.3.2 Mesh segmentation techniques

There are several algorithms for mesh segmentation that can be divided into categories according to their methodology. Some examples of these techniques can be found below.

Region grow; In this approach, new regions are expanding according to some rules. This set of rules determines whether the region will continue or stop growing. Vieira and Shimada in [34], present a region growth-based technique that automatically segments dense and noisy mesh models into regions that can be approximated by single surfaces. The algorithm first computes the noise levels and the curvature of the mesh model; it then partitions it into regions of connected vertices that are geometrically and topologically compatible with the input surface. This technique extends the existing region-growing algorithms and introduces methods for global noise estimation, threshold selection, and sharp edge detection.

Another example is presented in [35]; Sapidis and Besl perform a region growing segmentation technique that divides a mesh in regions that can be approximated by polynomial functions. Region growing is initialized with a seed region to which a polynomial is fitted. The region grows according to a distance and an orientation criterion; the approximating polynomial of that expanded region is then computed. The process continues until there can be no further expansion. An advantage of this algorithm is that the approximation surface of the new segmented part is constrained to a set of rules such as distance and orientation that the authors have set.

Clustering; in this approach segmentation is computed either by a dividing the mesh into a hierarchy of elements (faces, vertices) that are approximated using a geometric primitive, e.g. plane, cylinder, sphere or by using an iterative clustering approach where a k-means algorithm is applied [36],[37].

An example of hierarchical clustering can be found in [36]; in this work the authors have developed an hierarchical segmentation algorithm for triangle meshes that is based on fitting primitives such as plane, spheres and cylinders, belonging to an arbitrary set. The proposed method can automatically generate a binary tree of cluster elements, each of which is fitted with one of these primitives. Initially, each face represents a single cluster; for each new iteration, all the adjacent faces are considered as the next candidate, and the one that can be better approximated with one of the primitives will form a new single cluster. The approximation error is computed using the same metric for all the primitives, so that it can identify the most suitable primitive to approximate the set of faces in a cluster.

An example of iterative clustering using the k-mean algorithm is presented in [37]; the mesh segmentation is achieved using fuzzy clustering and cuts. The algorithm finds meaningful regions using a clustering algorithm while keeping the boundaries. It then uses a technique called fuzzy decomposition to find the exact boundaries that represent the object's features. To find fuzzy components, one condition must be fulfilled. Every face needs to belong to exactly one patch, and allow fuzzy membership. The algorithm starts by computing the distance for each face in the mesh. Each face is then associated with a probability value of belonging to another patch. Computing the decomposition can filter the probability values of each face thus identifying and constructing the exact boundaries of the mesh model.

Explicit Boundary Extraction; in this approach segmentation is produced by extracting the contours of a model. This technique has been used in [38, 39] where the authors apply a scissoring technique to extract the feature contours of the mesh based on its curvature. The scissoring operation is divided into 3 steps; the first step consists of extracting the feature contours by computing the minimum curvature value for each vertex of the mesh and converting them to closed contours. The contours are closed by using a geometric snake technique that consists of finding the shortest path using a combination of distance, normal and centricity criteria. These closed contours will represent the segmentation boundaries of the mesh. One of the main advantages of this approach is that it produces smooth and closed segmentation boundaries. More information about interactive segmentation using intelligent scissoring can be found in [40].

The work presented in [41] proposes a mesh segmentation algorithm by merging adjacent triangle pairs. The segmentation algorithm uses surface curvature to divide the model into one or more approximated planar patch regions, where each patch has a similar normal property. The algorithm initially compares the angle between the surface normals of the two adjacent triangles with a given error angle threshold. If the angle between the normals of the two triangles is close to the threshold value, the two triangles are separated into regions, and the edge between the two triangles becomes the boundary of the two regions. Additionally, the segmentation result can be controlled under a given error bound. An advantage of this technique is that it can be easily adapted to (LOD) Level of Detail techniques that can produce various mesh representation according to the resolution.

The technique used in this work can be classified as a hybrid approach of explicit boundary extraction segmentation and mesh simplification technique. The proposed approach is required to divide a given surface into boundary-based patches that will be represented using the PDE method. One of the main advantages of this technique is the ability of producing different LOD representations. Since a given mesh model can be described analytically using the PDE method, adjusting the u,v space parameters of each boundary patch can increase or decrease the surface resolution in a uniform or adaptive manner according to the end-user application. The new approach will be presented in more details in the following sections.

3.3.3 Automatic curve extraction of boundary based patches

The segmentation technique used in this work is required to divide a given mesh model to a set of triangular shaped patches based on the mesh curvature. Each patch will be later represented as a set of boundary curves that will be evaluated by the PDE method to produce PDE surface patches. Since mesh segmentation is not an easy task, the curve extraction technique is divided into several steps in order to have a better understanding of each process. This section contains the methodology required for producing an automatic curve extraction technique used to find a PDE-based surface representation of mesh models.

3.3.4 Methodology

The first step required for extracting a set of curves from a mesh model is to identify contour regions containing important features that need to be extracted. However, the technique for obtaining such results needs to be able to process

any given model regardless of its complexity. This is usually the most difficult step for most mesh segmentation techniques and there are several approaches to obtain this information (see Section 3.3.2 above). This work uses a mesh reduction technique to simplify a mesh model and use it as a guideline for extracting a set of template patches that can be used to obtain feature points from the original high resolution input mesh. The diagram in Figure 3.5 below describes the process of extracting the template patches using a simplified version of the input mesh. In this case, the extracted patches are represented as a set of template boundary curves connected to each other using the low resolution face connectivity. Note that each face of the simplified model will be converted into a curve containing 31 points. The number of points needs to be uniform for all the curve-sets representing the mesh geometry. This is due to uniform distribution of points between patches when computing the PDE method for a particular subdivision level. The process of converting faces to curves and obtaining the feature points from the original mesh will be described in a later section of this chapter.

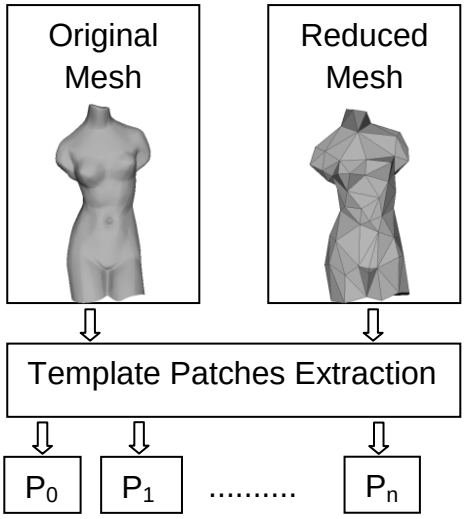


Figure 3. 5.Template patches extraction using mesh simplification.

3.3.4.1 Mesh simplification

Mesh simplification is a process used for reducing the number of faces used in the mesh model while keeping the overall shape and preserving the boundaries as much as possible. There are numerous algorithms proposed for mesh simplification that are dependent on the characteristics of the input mesh as well as on the needs of the application. These techniques can be categorised according to the approach, efficiency and quality of the resulting mesh. Some techniques can produce very good approximations but they lack in processing efficiency and are difficult to implement. Some other techniques can produce efficient approximations but they fail to preserve the topology and quality of the input mesh. Mesh simplification in this work is used to produce a desirable low resolution mesh that will be employed as a guideline for the template patches used in the extraction process. However, the results of the simplification are highly dependent on the complexity of the input mesh. To that extend, the reduction algorithm used in this work might not produce the best approximations for any given mesh model. A short overview of various mesh simplification algorithm and their applications will be covered in this section.

3.3.4.2 Mesh simplification techniques

Mesh simplification techniques can be grouped into two categories: local and global strategies [42]. Local simplification strategies [43] are usually quite ambitious; they simplify the mesh by repeating a process based on some local operator, whereas global strategies are applied to the input mesh as a whole.

A typical local simplification approach usually consists of an operation that when applied to the mesh, it processes small collections of elements (faces, vertices)

to produce the new simplified mesh. Determining which element should be processed is computed from an error or cost function that measures the cost-value relation the approximation would introduce after the operation. Additionally, this technique allows the user to specify the desired amount of elements to be removed.

Vertex decimation; an example of mesh simplification using decimation is presented in [44].The authors have developed a method that iteratively selects a vertex for removal, removes all adjacent faces, and re-tessellates the resulting hole. A vertex classification operation is used based on the face adjacency in order to determine the error associated with each decimation. During each iteration, a given vertex is flagged as a candidate for removal, if that a vertex meets the specified decimation criteria, then the vertex and all its adjacent faces are deleted. The resulting hole in the mesh is closed by applying a local triangulation scheme. This process is repeated until some target criteria such as percentage of reduction is met.

Edge Contraction; this is the most common mesh simplification operation. Edge contraction is the process of introducing a new vertex that is adjacent to all its neighbouring vertices and then deletes the endpoints of this edge and all their incident edges [45]. Edge contractions can alter the topology of a mesh, since repeatedly contracting all the edges around a hole will eventually close it. Additionally, edge contractions can be applied to edges containing non-manifold vertices [46]. It was original proposed in [47] as a method for surface reconstruction of unorganized points, and mesh simplification. The authors have introduced an energy minimization approach for solving the mesh optimization problem. The method produces simplified meshes with edges aligned along directions of low curvature, and vertices that are concentrated in areas of high

Gaussian curvature. The method can also recover sharp edges and corners since the energy does not penalize surfaces with sharp angles.

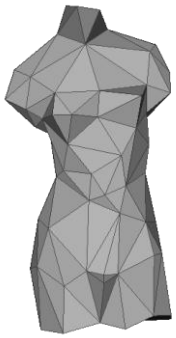
Another example of mesh simplification based on the use of Quadric Error Metrics is presented in [48]. Here, the authors have developed a simplification algorithm that produces efficiently high quality approximations using quadratic matrices for maintaining surface error approximations. The algorithm is based on the iterative contraction of vertex pairs which is a generalization of edge contraction algorithm. A vertex contraction is performed during a given iteration based on a cost function. This function is defined by computing the error approximation of each vertex. Additionally, the algorithm supports non-manifold [46] surface models. The algorithm can be summarized as follows:

- Compute the Quadratic matrices for all vertices.
- Select all valid pairs.
- Compute the optimal contraction target v for each valid pair (v1, v2). The new error becomes the cost for contracting that pair.
- Place all the pairs in an indexed list with the minimum cost pair at the top.
- Iteratively remove the pair (v1, v2) with the least cost from the list, contract this pair, and update the costs of all valid pairs involving v1.

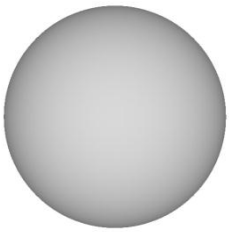
This algorithm was used for the mesh simplification process of the curve extraction technique. Its implementation was based in the Garland and Heckbert mesh simplification Quadric Error Metrics code [48]. Figure 3.6 below contains a mesh model of the Venus of Milo before and after mesh simplification process. The original mesh model, Figure 3.6 (a), contains 11,043 vertices and 21,977 faces. After mesh simplification the model was reduced to

106 vertices and 198 faces, Figure 3.6 (b). The polygon reduction was optimal for this mesh model due to its simplicity; the output mesh contains 198 faces that will be used for constructing 198 PDE-based surface patches. However, if the reduced mesh model contains faces with small areas and/or faces with long edges, it will produce errors during the curve extraction process. Additionally, there are cases where the results might need adjusting in order to extract the correct patch layout. In the case of Venus mesh model, Figure 3.6, the surface has been reduced more than 90 % percent from the original model to produce patches that can be converted into curve-sets. Complicated mesh models that contains a lot of curvature, non-manifold geometry or sharp edges require less reduction so that key feature points are maintained in the reduced version. It has been tested that a 20 percent reduction for complex models is enough to maintain its curvature without losing too much information. However, using a 20 percent reduction will produce a low resolution model with high number of faces that need to be converted to patches. Adjusting the percentage and the quality of the simplification will alter the final results.

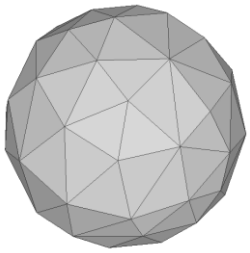
(a)



(b)



(c)



(d)

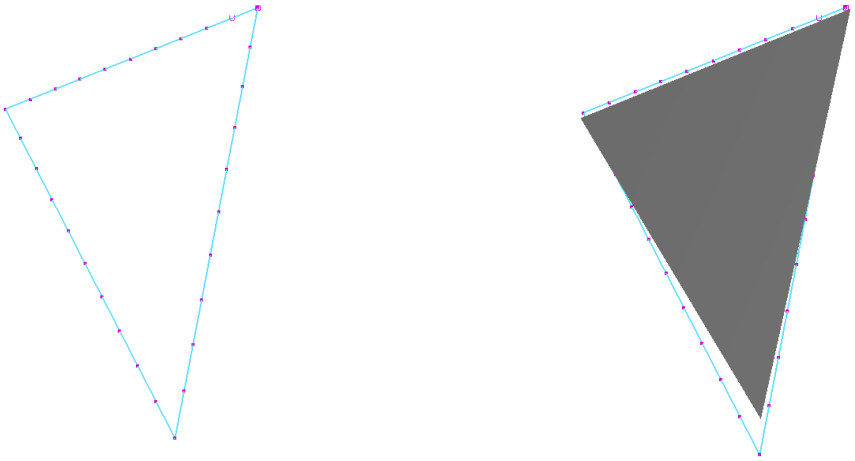
Figure 3. 6. Venus hi-res mesh model (11,043 vertices and 21,977 faces) (a) and low-res mesh model after mesh simplification (106 vertices and 198 faces) (b). Sphere hi-res mesh model (vertices 4.800, faces 9600) (c) and low-res mesh model after mesh simplification (vertices 77, face 150) (d).

3.3.5 Boundary patch segmentation

The low resolution model acquired from the mesh simplification process will be used as the guideline for the boundary patch extraction process. The mesh based reduction is used as a means for indentifying the contour features of the mesh. These features are contained in the low resolution mesh where the mesh simplification algorithm will respect them as constraints during the reduction process. The next step consists of converting the faces of the low resolution model to a set of curves that can be used to find the respective PDE surface. Each face of the reduced model is associated with the high resolution model This ensures that all regions of the original mesh will be included in the final model representation. This process is a simple operation that finds the distance from each of the vertices of the original mesh to a given face in the low resolution model, in order to find the closest point. A low resolution face is treated as a surface patch and the three points that each face contains are snapped to the closest point of the original mesh. This information will be useful for converting the face to curve points, where the three vertices of the face will be used to interpolate the in-between points of each edge.

3.3.5.1 Template boundary curve extraction

In order to satisfy the requirements inherent to the PDE method, the patches that represent the original high resolution model need to be converted into a set of boundary curves which the PDE methodology will use to produce a surface. Initially, a template curve needs to be extracted and used as a map for extracting the high resolution features, that are enclosed in that region, for that given patch. The template boundary curve is a curve consisting of 31 points, and is generated by linearly interpolating the two vertices of each edge contained in a given patch. The template curve can be represented as a triangular curve containing 10 points on each edge and one extra point used to close the curve, Figure 3.7 (a). Figure 3.7 (b), contains the template boundary curve with respect to the corresponding triangle in the low resolution mesh model. The triangle patch has been extracted from the low resolution model and used to map that region to the original mesh model. The two end points of each edge are repositioned during the low-to-high resolution patch association. The new end points have been snapped to the original mesh to ensure surface continuity during the feature extraction process. Lastly, the two edge points are linear interpolated to produce 10 points for each edge.



(a) (b)

Figure 3. 7.The template boundary curve associated with patch 1 of the Venus mesh model (a) and template curve and low resolution face (patch 1) of Venus model (b).

The new template curve represents now the patch for that region and will be used to map each control point to the high resolution model. This operation requires projecting each of the control points to the original mesh in order to find a point that lies on that surface. However, during this operation each control point needs to search every face of the original mesh until a ray-to-triangle hit has been achieved. This process can take a lot of processing time for an entire mesh. To that extend, a sub mesh partitioning technique is used to extract a mesh region that lies within the template boundary curve.

3.3.6 Sub-mesh partitioning

In this section a technique that partitions the original mesh into a number of mesh based regions is presented. Each of the new mesh regions is used to accelerate the curve extraction process and adjust the resolution of the region by subdividing it during the local feature extraction. The process starts by selecting a geometric primitive such as a box, triangle or circle to cover the patch surface. A point to primitive intersection test is then computed in order to

find which point of the original mesh is inside that region. If the point is inside the region, the position and the face connectivity are stored and used to construct the sub-mesh. When a patch is processed, the new mesh is stored indexed according to the patch it belongs. The process continues for each patch that is used to represent the high resolution mesh.

Three different primitives have been tested in order to produce better and more efficient results. Each primitive is first constructed to fit the patch in question. An intersection test is then computed for every point of the original mesh until all the points inside the primitive are found.

3.3.6.1 Point in box

A cubic box is the first primitive that was tested to identify points inside the patch region. Constructing a box aligned to the triangular patch is a simple operation that can be produced from finding the bounding box of the shape. A bounding box is constructed using the centroid and, the minimum and maximum points of the patch in question. The centroid can be found by averaging all the control points of the template boundary curve that belongs to that patch, as shown in Equation 3.1.

$$
\begin{aligned}
C_x &= (x_1 + x_2 + x_3 ..... + x_n) \ / \ n \\
C_y &= (y_1 + y_2 + y_3 ..... + y_n) \ / \ n \\
C_z &= (z_1 + z_2 + z_3 ..... + z_n) \ / \ n ,
\end{aligned}
\tag{3.1}
$$

where C_x, C_y and C_z are the Cartesian coordinates of the centroid and x_i, y_i and z_i are the number of points.

Once the centroid is found, the minimum and maximum points of the template curve are calculated and used to construct the bounding box of the patch. Equation 3.2 contains the formula for calculating the bounding box dimensions using the minimum and maximum points.

$$\begin{aligned} \text{width} &= \text{max.x} - \text{min.x} \\ \text{height} &= \text{max.y} - \text{min.y} \\ \text{length} &= \text{max.z} - \text{min.z}, \end{aligned} \tag{3.2}$$

The process then continues by testing all vertices of the original mesh model against the bounding box. Point inside a box intersection is a simple operation that can be tested using the code in Figure 3.8 below. Additionally, a tolerance value can be added to the minimum and maximum value to expand the bounding box in cases where the boundaries of the extracted mesh do not cover the entire patch region.

```
// If point is inside bounding box limits
if ((point.x >= min.x) && (point.x <= max.x) &&
    (point.y >= min.y) && (point.y <= max.y) &&
    (point.z >= min.z) && (point.z <= max.z))
{
     // Point found
}else
     continue;
```

Figure 3. 8. Pseudo code for Point inside Box intersection test.

Using a box as intersection primitive offers computation efficiency during the construction of such primitive and the intersection process. However, the extracted sub-mesh contains a lot of extra data that cannot be used. This is due to triangle-to-box fitting; a lot of points that are not inside the triangle but are inside the box will pass the intersection test and will be included in the sub-mesh. Figure 3.9, contains the extracted mesh using the point to box intersection approach.

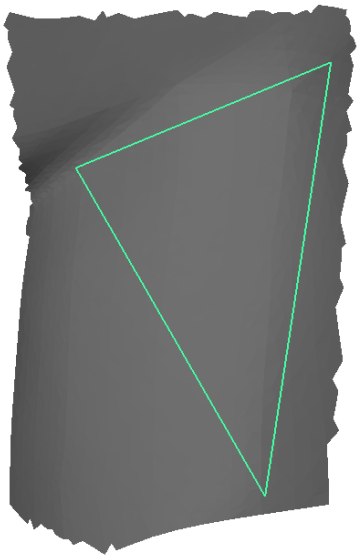


Figure 3. 9.Sub-mesh extracted using the point in box intersection.

3.3.6.2 Point in circle

The next primitive to test is a circle. In order to construct such a primitive, a circle or circumcircle needs to be computed. This circle needs to pass through the three extreme points of the template boundary curve. However, constructing a circle that fits three points is less computational efficient compared to constructing a box. Constructing a circle requires a point and a radius. In this case the circumcenter of the triangle that is found using the three extreme points; whereas the diameter of the circle is computed from the length of any side of the triangle. The circumcenter of a triangle can be found as the intersection of the three perpendicular bisectors. The perpendicular bisectors are calculated from the cross product between each of the edges and the normal vector of the triangle. Using a line intersection test, the point where of these vectors intersect is the circumcenter of the triangle, Figure 3.10 shows the pseudo code used for finding the required circle.

```
// Distance between current point and circumcenter
Vector d = point -  circumcenter;
Float magnitute = sqr(d.x * d.x + d.y * d.y + d.z * d.z);
if(magnitute <= radius)
{
     // Point found
}else
     continue;
```

Figure 3. 10.Pseudo code used for finding the Point inside circle intersection test.

The advantage of using a circle as a mesh extraction primitive is the simplicity of the point in circle intersection test. Since it is constructed to fit the triangular patch, the test whether a point is inside the circle can be found by comparing the magnitude of the distance between the point in question and the circumcenter, with the radius of the circle, Figure 3.10. On the other hand, constructing a circle requires a lot more computations while the results can still include a lot of unnecessary data. Figure 3.11, contains the circular mesh extracted using this approach; many areas of the circular mesh will not be used during the projection test since they are outside the triangle region.

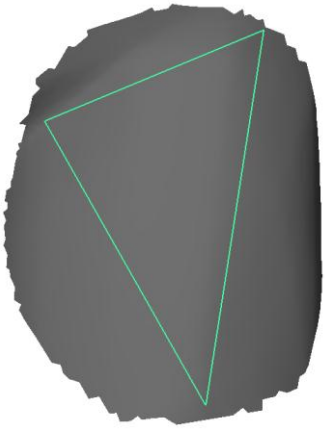


Figure 3. 11.Circular sub-mesh extracted using the point in circle intersection.

3.3.6.3 Point in triangle

The last primitive to perform intersection test is the triangle. The triangle is the best candidate for the sub mesh partitioning process, since constructing such primitive does not require any additional estimation. The process is initialized by defining a new triangle that consists of three extreme points taken from the template boundary curve. Prior to perform the intersection test, the triangle might require scaling since some points in the region might fail during the test.

A scaling factor of 0.5 is usually enough to include all the necessary points around the original triangle area. The process then continues by testing all the points in the original mesh using a point in triangle intersection test. There are various methods for determining if a point is inside a triangle, more information can be found in [49]. A common method of testing whether a point intersects a triangle is to intersect a ray with the plane defined by the triangle and then determine whether the intersection point lying on the plane is inside the triangle or not. In order to identify in which side of the line a point is on, is computed using the cross product of the two vectors that are defined between two point of the edge and the point in question is computed. The direction of the cross product follows the right-hand rule. If the dot product of at least one vector points in different direction from the rest then the point is outside of the triangle. If the dot product of all vectors point to the same direction, the point is inside. This is very simple and effective method that requires calculating the cross and dot product three times or less per triangle. Note that the direction of the cross product is dependent on the face orientation. In either case this can be easily adapted by changing the condition from negative to positive. If the dot product of all vectors is positive then the face is clockwise otherwise it is anticlockwise. Figure 3.12 shows an example code for performing the point in triangle intersection test. For each edge, two vectors are constructed between the edge points and the point in question. The dot product of each of the perpendicular vectors is then compared to determine if it points in the same direction for all edges. A vertex is rejected if it points in the opposite direction.

```
   Vector pV2 =  point1 – Point;      // edge 3 point 1
   Vector pV0 =  point0 – Point;      // edge 3 point 0
   Vector q2 = Cross product(pV2, pV0);

   Foreach edge of the Triangle
   {
        Vector pV0 =  point0 – Point;    // edge i point 0
        Vector pV1 =  point1 - Point;    // edge i point 1
       // calculate cross product between the 2 vectors
        Vector q = Cross product(pV0, pV1);
       //Compare each perpendicular vector
         if (q2.DotProduct(q) < 0)

             return false;//Point is outside triangle
}
```

Figure 3. 12.Pseudo code for finding the Point inside triangle test.

This primitive will be used to extract all the required data for the feature extraction process. Using a triangle offers great advantages in the sub mesh extraction process. The construction of such primitive requires no calculation at all, whereas the intersection test is more efficient and accurate compared with the previous techniques. Figure 3.13, shows the triangular mesh extracted using this approach; most of the extracted data belong in the corresponding patch area, only a few set of points will be ignored since they are outside of the triangular region. However this can be controlled by adjusting the scaling factor during the point in triangle operation.

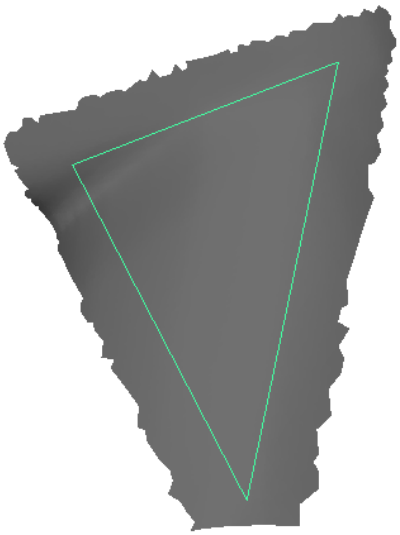


Figure 3. 13.Triangular sub-mesh extracted using the point in triangle intersection.

3.4 Feature extraction

The procedure of finding a sub mesh region for each of the patches representing the input mesh will accelerate the overall process, while being able to locally adjust the sub mesh surface resolution; it can also increase the quality of a particular patch. The next step consists of transferring features from the original mesh to the template boundary curves. As mentioned before, the template boundary curve represents the boundary limits of a patch obtained from the low resolution mesh model and its is used as a guideline for extracting features that lie inside that region. As seen in Figure 3.13 above, the curve that is inside the triangular mesh region represents the template boundary curve. At this point, the template curve consists of a set of points forming a planar triangular face. The feature extraction process will use the control points of each template curve to project them to the sub mesh region they belong. Projecting a point to a triangle is a technique used in ray tracing [49] for finding the intersection point between a ray and a triangle. The use of a the sub region of

the surface will accelerate the process since only the triangles that are inside the patch region will be tested to see whether they are valid projection triangles or not, rather than iterating though the entire mesh.

3.4.1 Feature extraction using raytracing

Ray tracing is a global illumination technique used for rendering. During ray tracing, a ray is usually generated from the position of the eyes back to the scene. All the rays are tested against all objects in the scene to determine whether they intersect any object or not. If an intersection is found, ray tracing can handle shadows, reflections, and/or texture mapping according to the rendering requirements. Ray tracing has many uses in computer graphics as well as in other fields, for example it can be used for extracting data in acceleration structures [50], usually achieved using ray traversal [51]. In this work, ray tracing has been used for projecting a point to a surface during the feature extraction process. The process starts by constructing a ray for the first control point of the template boundary curve. The position of that point is used as the starting position of the ray, however the direction of the ray plays very important role in maintaining the correct projection direction. Once a ray is constructed, it is tested against all faces in the current sub mesh to find the point of intersection. Once the intersection point is found, it replaces the original control point of the template curve and continues to the next point until they are all projected. This technique guarantees that every new point will lie on the surface of the original mesh model.

Finding the intersection point consists of a two step operation, initially the ray is tested whether it hits the triangle using the technique discussed in Section 3. If

the ray hits the triangle, it uses barycentric coordinates to find the exact location in the three-dimensional space.

As mentioned before, choosing the correct ray direction is a process that can affect the quality as well as the accuracy of the resulting points. A quick solution would be to use the face normal for the ray's direction; however this ray does not always guarantee that it will hit the correct triangle. Moreover, if the face contains edges that are boundary to a hole in the mesh, the face normal direction will not hit any elements. Three different approaches were tested for obtaining the most appropriate ray direction. The first test was carried out using the face normal direction for every point in the curve. The results using this vector are satisfactory but there are many cases where the resulting points of intersection for many patches across the mesh are not the best selection. Additionally, this approach requires aligning all the edges per curve according to the face connectivity after the extractions is finished.

The next ray test consists of using the edge normal. Edge normals are calculated by averaging the normal of each neighbouring face with the current face normal per edge. Figure 3.14 below shows an example of using the edge normal for ray direction. The face normal d_1 from face A is averaged using the normal d_2 of the adjacent face B. The resulting normal d_3 will be used as the direction for every point of that edge.

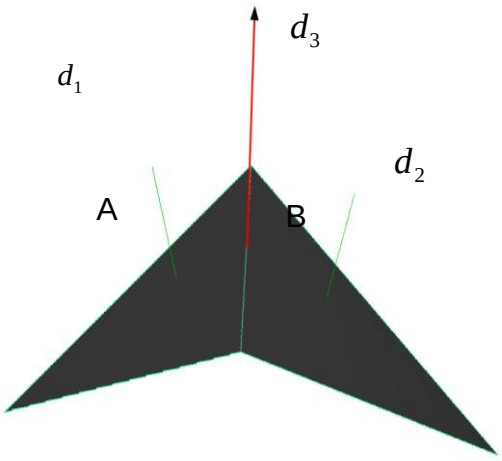


Figure 3. 14.Edge normals for ray direction. The vector d_3 denotes the ray direction

Each edge normal is computed separately using its adjacent face neighbour; if an edge is boundary and does not contain any neighbours then a different direction must be found. This limitation generates problems since the input mesh can include any form of configuration. And meshes that fall into the above category will fail.

Lastly, the ray direction used for feature extraction is based on the vertex normals of each patch. In this case, the vertex normals between the two end points of each edge are used to linearly interpolate the new position. Each new interpolated vector is used as the ray direction for the control point that belongs to. Figure 3.15, below shows the interpolated vectors for each control point between the two edge end points; each vector is used as an independed ray. This approach guarantees optimal results even in cases where the mesh contains holes or sharp edges between the patch edges. Additionally, every new projected point will be shared correctly with the neighbouring patch edge. This will ensure surface continuity when calculating the PDE surface for each patch.

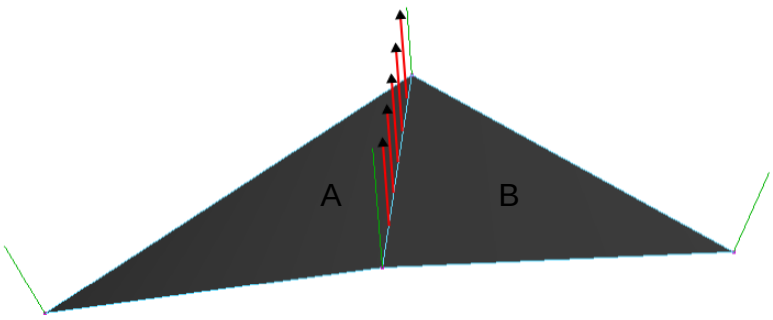


Figure 3. 15.Vertex normals for ray direction.

3.4.2 Acceleration constraints

Ray tracing is a demanding operation that requires a lot of processing time; although the sub mesh surface is used to accelerate the process for each patch, this process is highly dependent on the total number of patches that are used to represent the original geometry. There are cases where the original input model can be correctly characterized using 1000 or more patches. In such cases, the curve extraction operation is expected to take hours to calculate all the necessary patches. Additionally, every patch contains one template boundary curve with 31 points, once the process successfully projects all the point to the surface, the operation continues with the generation of three additional curves that will cover the inner areas of each patch. Each inner curve is created by scaling the template boundary curve three times; every new inner curve will be projected to the surface using the same procedure. To that extend, every patch contains 4 curves with 31 points each that need to be projected to the surface. If the number of patches is 1000, the total number of test for 124 points per patch is 124.000; which is the total number of projections needed. Additionally, every sub mesh contains extra vertex information that will be tested 4 times per curve; this is due to the fact that if for example a sub mesh contains 1000 faces only 124 faces will be used for the patch in question.

However, there are several techniques that can accelerate such process by indexing all the necessary data in a spatial grid. A spatial grid can be described as a grid that is partitioning itself into regions of cells or voxels. Each voxel contains a list of objects that are associated with it. Extracting information from such data structure can be achieved using grid traversal [51]. During grid traversal, a ray is shot towards the grid. If it hits any objects in the starting voxel list, the intersections are sorted and the closest one is returned. If no intersection is found in the current voxel, the next neighbouring cell is searched. The process continues until either an intersection is found or the entire grid has been traversed. There are two popular space partition techniques: octrees [50], voxels in the grid contain different sizes, and constant size voxel partitioning [52]. Implementing such a scheme will help accelerate the ray tracing process; every point that needs to be projected will be tested with all faces that are associated with the cell that intersects the ray, rather than testing all the faces in the sub mesh. The number of cells in the x, y and z direction also plays an important role in increasing or decreasing the calculation time. To that extend, the bigger the cell is, the smaller the number of data that are associated is. This technique is not used in the current curve extraction version and is proposed as an add-on to help accelerate the whole process for future work.

3.4.2.1 Surface computation

The final step of the curve extraction technique consists of evaluating the extracted data for computing the PDE surface. Every patch contains 4 curves projected on the original surface that satisfy the boundary requirements for solving the PDE method. Figure 3.16 (a), shows the template boundary curve

projected to the sub mesh surface associated with the first patch of the Venus mesh model. Figure 3.16 (b), contains all the curves required for computing the PDE method. The three inner curves have been created by scaling the template boundary curve and projecting it to the surface. The 4^{th} curve contains only one point, this is the centroid of the boundary curve projected to the surface, and is used to close the surface. Figure 3.16 (c) and (d) consist of the generated surface using the PDE method at subdivision level 3. Note that being able to represent a surface analytically, offer us the advantages of parametric surfaces such dynamically adjusting its resolution. Moreover, the surface can be re-computed to increase or decrease its resolution from subdivision level 0 to theoretically any given number. The application is configured to calculate 9 subdivision levels, since the total number of vertices at this level increases to 1.500 and 2.200 faces per patch. Usually most of the details of the surface are reproduced between levels 4 and 5; a higher number of subdivisions add only extra points and faces.

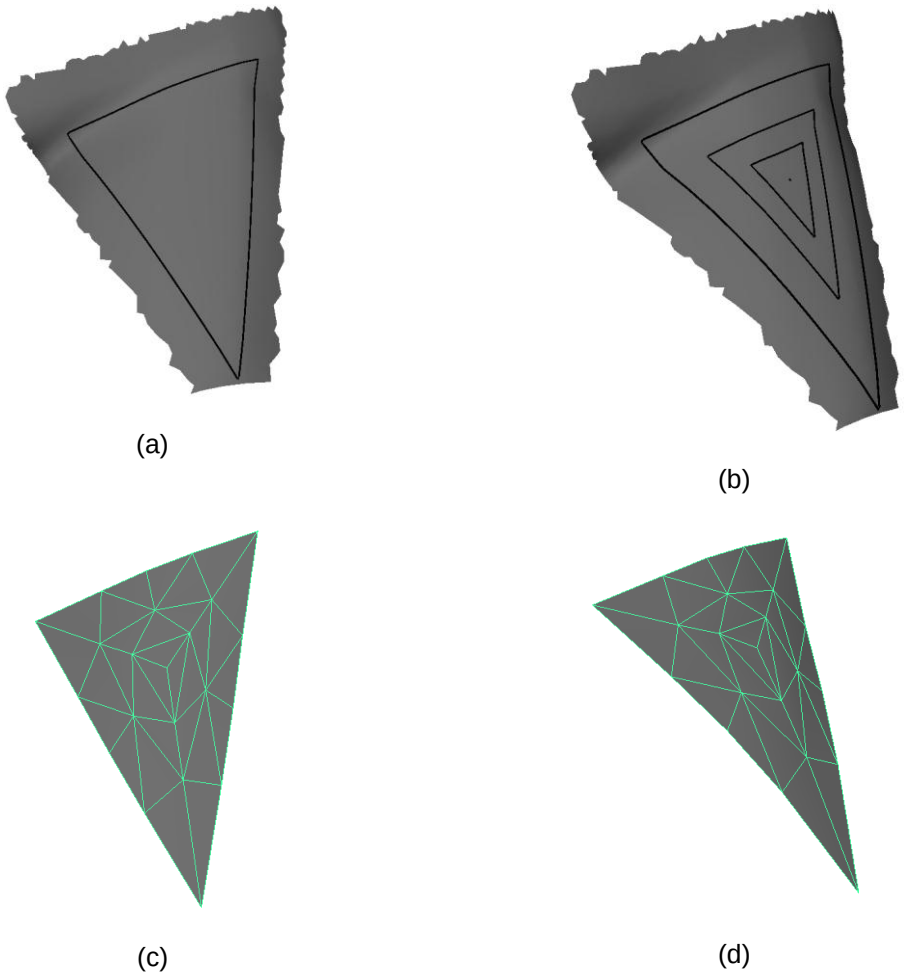


Figure 3. 16.Final projected boundary curve (a). The final projected curve-set required to produce the PDE surface (b). Front and side view of the PDE surface at subdivision level 3 (c) and (d).

3.5 Examples

This section contains examples created during the automatic curve extraction process. Figure 3.17 (a) below contains the template boundary curves that represent the front side of the Venus model. The curves have been extracted from the reduced version of the Venus Model; each template boundary curve is a flat triangular curve consisting 31 points. In total there are 198 PDE patches, each mapped to a face obtained from the low resolution reduced model. Figure

3.17 (b), shows the PDE patches after the feature extraction process. Each boundary curve has been projected to the original model surface using the ray tracing techniques discussed in previous section. Finally, Figure 3.17 (c) shows the PDE patches containing the inner curves. The inner and boundary curves in each PDE patch are used as the boundary conditions required for computing the PDE method.

(a) (b) (c)

Figure 3. 17.The template boundary PDE patches that represent the Venus model (a) .The final boundary PDE patches after the feature extraction process (b). PDE patches containing inner curves (c).

Figure 3.18 contains the PDE surfaces obtained in different subdivision levels. Figure 3.18 (a) shows the original Venus mesh model from where the PDE boundary data came from. The rest of the figures show the new PDE surface consisting of 198 patches for different subdivision levels. Subdivision level 0 consists only from 3 points that form a triangle patch. At this level the model is the same as the low resolution model; see table 3.1 below for total number of vertices and faces comparison. Figure 3.18 (c) contains the PDE model at subdivision level 2; at this level most of the features of the original model have been obtained.

Model	Elements	Original	Subdivision 0	Subdivision 2
Venus	Vertices	11.043	594	1.980
	Faces	21.977	198	2.376

Table 3. 1. Total number of elements for each subdivision level compared to original Venus model

Finally Figure 3.18 (d) contains the PDE surface in subdivision level 4, this level consist of 4356 vertices and 5940 faces. At this subdivision level some problems begin to appear in the PDE surface, this is due to the number of points that are concentrated in triangular patches with small areas. As a solution to this, a different triangulation scheme can be used to improve the distribution of points across each PDE patch. Additionally, problems might appear from averaging the normals between the patches. Note that there are still features in the original model that do not appear in the PDE representation; this is usually

caused by the automatic extraction process since only a small selection of points are extracted to represent a region in the original mesh model.

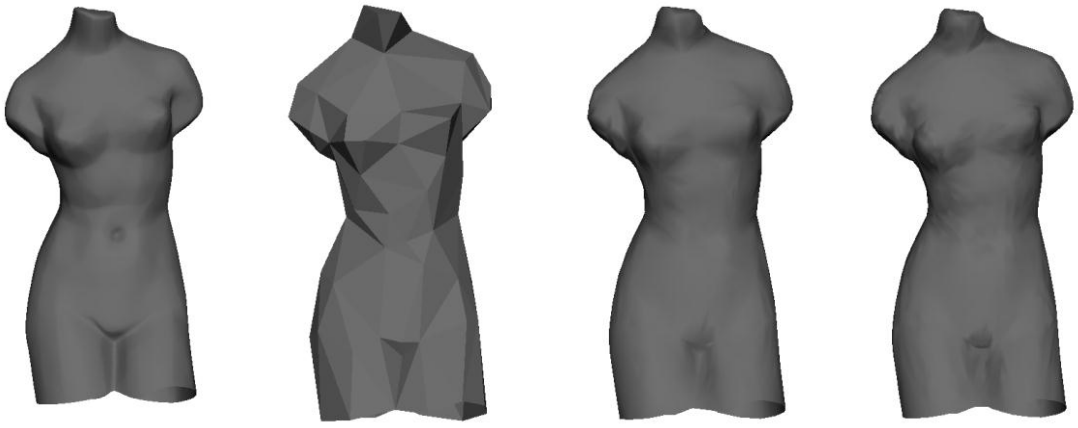
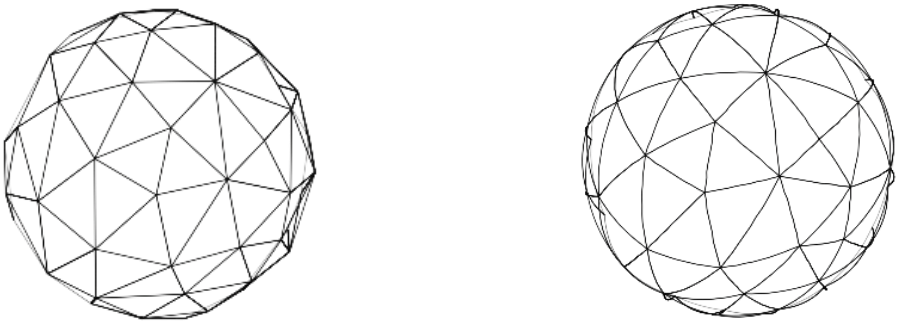


Figure 3. 18. The original Venus mesh model (a).
The complete Venus PDE model in subdivision level 0 (b).
Subdivision level 2 (c). Subdivision level 4 (d).

Figure 3.19 below shows the curve patches used to compute a sphere model. Figure 3.19 (a) contains the template boundary patches extracted from the low resolution sphere model, whereas Figure 3.19 (b) contains the final PDE patches after the feature extraction process. A sphere is a good example for the curve extraction technique since there are not any feature characteristics inside each patch. The Figures consists of the final PDE surface in two different subdivision levels compared with the original sphere model. As seen in Figures 3.20 (a) and (c) the original model and the PDE representation at level 2 are almost the same. Subdivision level 0 is the most basic configuration and is identical to the low resolution sphere model.



(a) (b)

Figure 3. 19. The complete curve set for the template boundary curves that represents sphere model (a). The final boundary curves after the feature extraction process (b).

(a) (b) (c)

Figure 3. 20. Original Sphere mesh model (a).
The PDE surface representation on Sphere at subdivision level 0 (b).
Subdivision level 2 (c).

Table 3.2 below contains the total number of vertices and faces for subdivision level 0 and 2. The PDE sphere at level 2 looks like the original model, while using a lot less number of vertices and faces.

Model	Elements	Original	Subdivision 0	Subdivision 2
Sphere	Vertices	4.800	450	1.500
	Faces	9600	150	1.800

Table 3. 2. Total number of elements for each subdivision level compared to original sphere model.

3.6 Conclusions

The automatic curve extraction technique presented in this section is a new approach of mesh simplification that has been used to obtain the boundary conditions required for calculating the PDE method. The technique starts by dividing the input mesh model into a set of boundary curve-based patches. The mesh is reduced using a mesh simplification technique until it reaches a satisfactory level of quality and number of faces. Each face is then converted to a template boundary patch consisting of 31 control points. These points are then projected to the original mesh in order to extract the features that lie within that region. Once a boundary curve is complete, the process continues by generating 3 additional inner curves for that particular patch; 4 curves in total per patch are used to calculate the corresponding PDE surface. Once a given model can be described as a set of curves, it can be reconstructed using PDE surfaces over a given level in real time. This offers great advantage in environments where the LOD controls the resolution of the model determined by the distance from the user to the object. The PDE data required for constructing a surface are much smaller in size compared to any optimized mesh model. Only a small set of curves containing 31 points is enough to represent a given mesh model consisting of thousands vertices and faces.

However since the PDE method is an approximation technique, some features might get lost during the evaluation of the surface. It is also guaranteed that some features will be lost during the automatic curve extraction process; this is due to various operations (mesh reduction, ray tracing) that take place in the original mesh model. Future work can be undertaken in optimizing and refining the results of the curve extraction process. Additionally, a web based implementation for computing PDE surfaces in web based visualization environments, can be used to accelerate the transmission time between client/server communications.

Chapter 4: Manipulation of PDE surfaces

4.1 Introduction

In the last chapter we examined various techniques for curve generation that satisfy the boundary requirements of the PDE method for various applications. Without the boundary data it would not be possible to generate any form of parametric PDE surfaces. Here we explore the PDE method in presenting two different applications capable of designing and manipulating parametric surfaces in real time.

4.2 A toolkit to design and manipulate aircraft geometry

4.2.1 Introduction

Plane Designer is an application designed and used for geometric modelling using PDE based airplanes written in Visual C++ and OpenGL .The construction of airplane surfaces in this work has been generated using the PDE method. Unlike spline techniques, the PDE method can produce complex surfaces in terms of a small set of design variables or parameters. The shape of the surface is defined through boundary curves and a small set of design parameters, taking a boundary value approach to the problem of surface design. This approach allows the design system to be extended so that the functionality of the object can be taken into account at an early stage of the design process. Additionally, it has been previously shown in chapter 2.1.2 that the use of the PDE method [25] can significantly reduce the computational cost associated with the process of designing and optimising the performance of either a given airplane surface configuration or specific components such as wings or fuselage [53, 54]. Here, the optimisation of the airplane geometry takes

advantage of a small set of design parameters required by the PDE method to define complex geometry. Thus, using such set of geometry parameters, it is possible to optimise the shape of an aircraft automatically in a reasonable time scale. To that extend, it opens the possibility to automatically compute a shape which minimises lift or drag.

The aim of this work is to discuss the advantages of using the PDE method in modelling airplanes in contrast with the use of other existing methods and to show how airplane surfaces with more general boundary conditions can be constructed and manipulated in real time. Additionally, it is demonstrated that this technique is capable of representing and manipulating an already existing geometric model of an airplane without any prior knowledge on the part of the designer regarding the mathematical details of the PDE method itself. For the purpose of explaining the use of the PDE method in airplane design, we will use examples illustrating the various designs, transformations and modifications of airplane models. All the examples presented throughout this work have been created real time from the PDE Airplane designer application.

4.2.1.1 Background

One of the major tasks in the design phase of a new aircraft is the definition of its configuration along with the main geometric characteristics. Present CAD packages provide various tools for the parametric representation of complex geometries and surface definition is usually accomplished by utilizing such CAD systems. For an aerodynamic early phase conceptual design, a step before the application of CAD is needed, for example a toolbox that will produce generic and parameterized aerodynamic surfaces, which will take into account the

special needs and constraints for the conceptual design of an aircraft [55]. Besides the well known general CAD packages, very few are specialized in aircraft design. Some of them are highlighted below.

Klein and Sobieczky [56, 57] present examples of aerodynamic design of high speed airfoils and wings which is carried out by their Genetic Algorithm software. They use explicit functions to describe curves needed in the design of aircraft surfaces. Their goal is achieved by establishing a flexible input data generator for both direct and inverse design. The geometry and flow quality are modelled by a set of analytical functions with parameterized input. For communication purposes with CAD packages, the resulting surface is interpolated using NURBS surfaces.

A surface generation software named Ge.P.A.S. (Generic Parameterized Aircraft Surface) developed by Sarakinos and Valakos [58] is also designed for the construction of aerodynamic aircraft surfaces. The surface generation procedure is parameterized and different aircraft configurations can be produced interactively. The surface generation procedure is based on the use of NURBS curves and surfaces. Fuselage type surfaces are constructed inside a scalable reference volume, where common basic curves in successive parallel planes are transformed to form the corresponding cross sections. Wing type surfaces are constructed in a more straightforward manner, using standard formulations for wing geometry definition and a database of wing sections to select from.

RAGE [59] is another package that has been designed for the generation of aerodynamic models, where central parameterized geometry definition has been used. It is used at the preliminary stage of aircraft design for preliminary

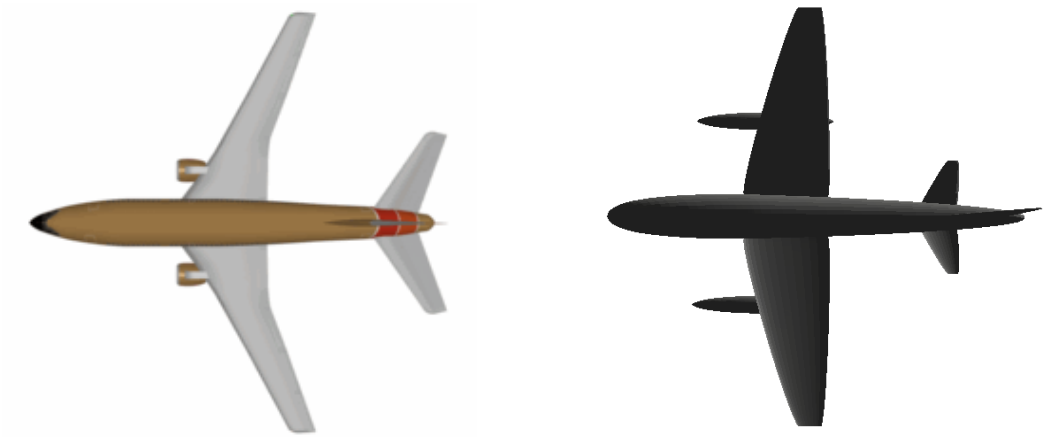
parametric studies and optimization. Designers can develop aircraft geometries that vary from very simple to quite detailed configurations for analysis with an assortment of computational aerodynamics tools in a very fast manner. Aircrafts are designed with the use of fuselage, wings and engine components which are available in the software. Each component is generated by a number of subcomponents that define the geometry. RAGE is able to output files that are compatible with several aerodynamic analysis codes and is compatible with most of the major CAD packages.

4.2.3 Creating an aircraft

The airplane design application discussed in this section requires creating and manipulating PDE-based aircraft geometry. The basic idea in creating any surface patch or an airplane shape using the PDE method is to define the boundary conditions appropriately and seek the solution to its corresponding PDE for generating the associated surface [21]. The aircraft design implementation process is divided in three parts, the fuselage, wings and tails geometry. Each of these parts contains similar properties but different characteristics that can be adjusted interactively by the user. Each airplane component is controlled by a set of parametric curves used as the boundary requirements for evaluating the PDE method. Visualization and user interaction are handled from the (Open Graphics Library) OpenGL API [9], which is a standard specification defining a cross-language API used for 2D and 3D graphics. One of its basic operations is to accept primitives such as points or polygons and convert them into pixels though the OpenGL graphics pipeline. For selection and picking of various primitives OpenGL provides a selection

rendering mode. However, other techniques exists to offer more flexibility, here every primitive is rendered in a unique colour while in selection mode. Then each single pixel colour, under the current mouse location is examined to determine whether that primitive is selected.

This work is an attempt to integrate and utilize the PDE method as a toolkit in the design and manipulation of aircraft geometry. For more complex configurations more aircraft parts need to be included in the system to improve the overall representation of the final geometry. Images of existing airplane geometry have been used as a guideline to understand various model designs. Figure 4.1(a) below shows the main parts of a Boeing 737 used in the current application. Figure 4.1 (b) contains a generic PDE surface aircraft configuration designed in the current application. The creation and manipulation of each body part will be described in the following sections.



(a) (b)

Figure 4. 1. Template shape of Boeing 737. (a)
A representation of a generic PDE-based surface aircraft configuration. (b)

4.2.3.1 The fuselage

The first step in the aircraft design process is the fuselage creation. The fuselage is the main body section of an airplane that holds crew and passengers or cargo. The interactive platform allows the user to select the number of curves that represent the fuselage, the distance between them and the length of the two diameters, minor and major axis, of each of the curves, as shown in Figure 4.2 (a). These parameters serve as the initial configuration of the fuselage shape. The initial number of curves will also play an important role in the construction of the PDE surface since a single 4^{th} order PDE surface requires four curves; the first and last curves are the boundary conditions describing the position of the surface whereas the rest of the curves determine the overall shape of the surface. The fuselage surface in each configuration is designed with two PDE surface patches blended together. For this case the PDE method requires seven curves as input. The first four curves define the first PDE surface or patch, while the last curve of the first patch combined with the rest of the curves will define the second PDE patch, as shown in Figure 4.2 (a). Although the current version of the application is capable of producing up to two blended PDE patches for each aircraft part, more patches can be implemented.

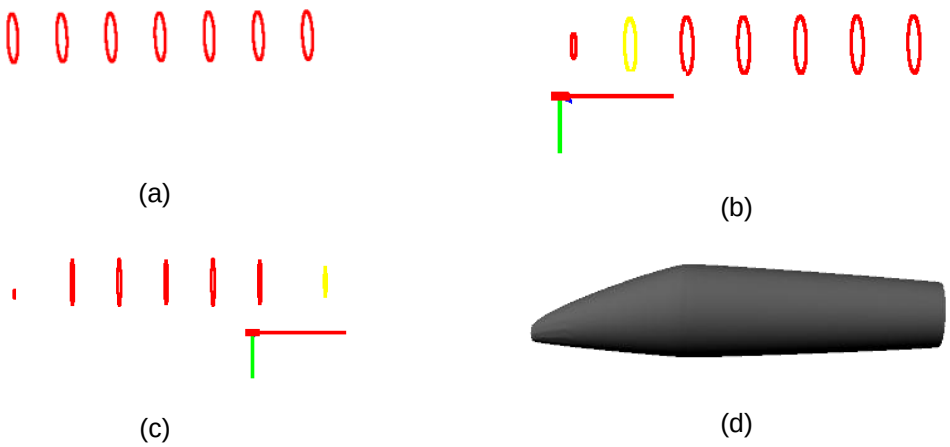


Figure 4. 2The parametric fuselage component construction process. Initial curve configuration for the fuselage (a). Interactive manipulation of the fuselage (b) (c).
PDE-based surface representing the fuselage (d).

After these curves are inserted in the system, a cylindrical shaped template surface is constructed as the initial fuselage (Figure 4.2 (d)). Figure 4.2 contains the steps for designing the fuselage. Figure 4.2 (a) shows the initial cylindrical template used for manipulation. Simple transformations such as translation and scaling are applied to each curve separately to adjust the overall shape, Figure 4.2 (b) and (c). Additionally, the system offers various tools for interactive design; these include general transforms that can be applied to each curve in design mode. The user can choose a transformation method and select a curve from the rendering window; a transformation axis that controls the direction of the movement over the x, y and z coordinates is displayed to guarantee better manipulation. When the fuselage has reached the desired shape, a surface generation function is available to calculate the PDE surface for the current curve set configuration. Figure 4.2 (d), consists of two PDE surface patches blended together in order to increase the shape control and the representation of the final geometry.

4.2.3.2. Constructing the wings geometry

The next step in the aircraft design process is the construction of the wing configuration. The wing component construction follows the same steps as with the fuselage generation process. An additional step is required for constructing the initial curve-set on the fuselage surface. The first curve of the template wing curve-set will be projected automatically on insertion to the fuselage surface in a way that the resulting surface will be blended to it [17].

First, the exact position of the surface that will be blended needs to be examined, as illustrated in Figure 4.3 (c). Any chosen point in the (u,v) parametric space will be mapped to a point on the surface. Thus, by creating a curve on the parametric domain, points on the surface can be identified and generated on the required curve. Note that any planar curve drawn in the (u,v) parametric space will be guaranteed to lie on the surface. The new curve can be manipulated by the user, where simple transformations such as scale and translate are applied directly to the (u,v) space where the curve is projected. Every time the curve is adjusted, the transformations will be applied to the projected curve. This technique ensures that the wing geometry will always lie on the target surface, which in this case is the fuselage, blending together the two surfaces in question.

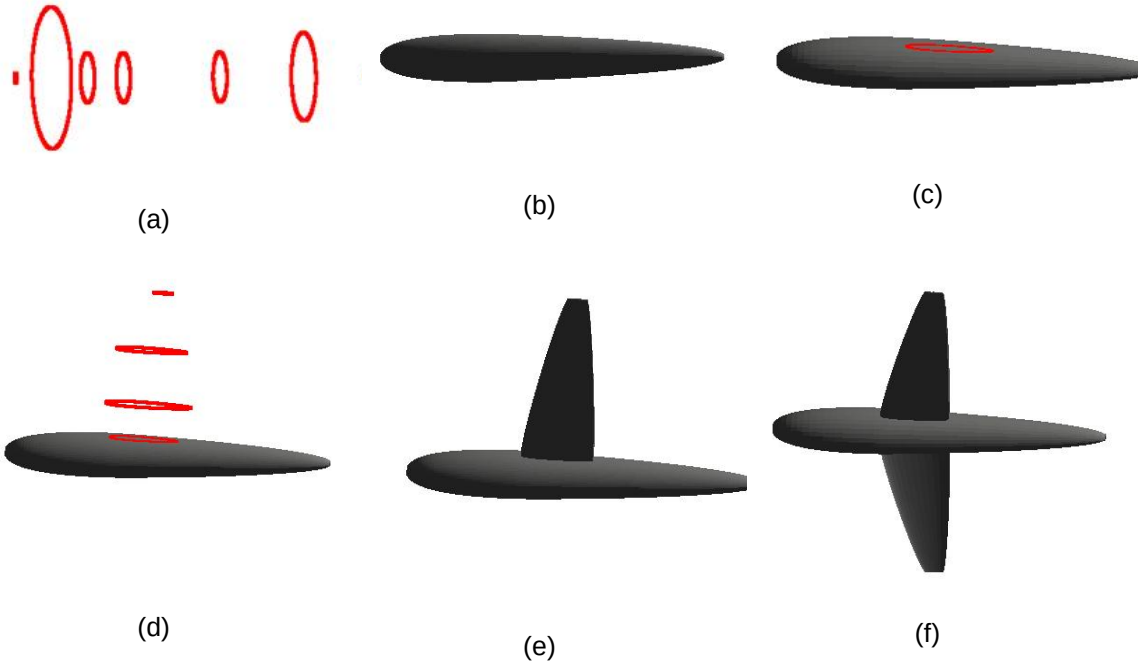
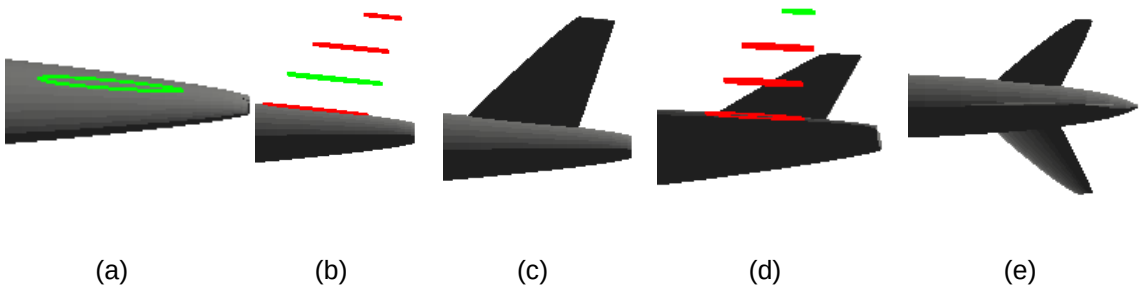


Figure 4. 3. Construction of a generic aircraft shape. Fuselage curves (a). Two patches fuselage object (b).The initial curve for the wing (c). The generating curves of the wing (d). Wing and fuselage objects blended (e). The final basic shape of the airplane (f).

Once the initial position of the curve, Figure 4.3 (c), is defined according to the aircraft model requirements, the process continues with the insertion of the remaining wing curves aligned in that position. This set of curves can be individually transformed to achieve the required shape for the wing. Figure 4.3 (d) shows a pre-configured set of curves representing the wing part of the airplane. When the wing curve configuration is complete, the system can produce the resulting PDE surface representation for the given boundary conditions as seen in Figure 4.3 (e). In order to complete the construction of the wing geometry, the new PDE surface needs to be reflected to the opposite direction (Figure 4.3 (f)). This can be achieved by mirroring all the vertices of the current mesh to the desired wing axis direction. These new data can be used to store and display the new of the wing surface.

4.2.3.3 The PDE surface representation of the tail

The last step, depending on the aircraft configuration in question, involves the design of the rear tail wings. The process is the same as described in previous section; a curve will be projected on the fuselage surface to define the initial position of the tail wing shape, Figure 4.4 (a). A new curve-set aligned to that blended curve will be inserted and manipulated in order to achieve the desired shape. Figure 4.4 below contains all the necessary steps for the rear tail wing construction process. Steps a, b and c are required to design the vertical tail or vertical stabilizer, whereas steps d and e contains the construction of the horizontal stabilizers of the aircraft design.



(a) (b) (c) (d) (e)

Figure 4. 4. Construction of the PDE–based rear tail wing component. Creating the initial position of the wing part (a). Initializing and manipulating curve-set representing the wing (b). The PDE-based representation of the wing surface (c). Designing the side wings (d)(e).

4.2.4 Applying transformations

Surface manipulation is carried out intuitively in an interactive environment by using a set of parameters which are associated with the boundary conditions [60]. A curve can be individually selected and manipulated within that environment by applying various transformations. Each transformation, such as translation, scaling or rotation, is based on parameters through which the user

is able to change the original shape of the curve to the desired shape by the user. To that extend, mouse coordinates are used and the parameter is updated accordingly in the axis the user has selected. Figure 4.5 shows the transformation axis that controls the transformation direction of selected curves. By selecting a transformation from the menu and a curve, an axis appears that controls the x , y and z coordinates of the chosen curve in the three-dimensional space. The transformations used in this chapter are all examples of affine transformations. One of the essential properties of such transformations that need to be noted is that the curvature of an object will remain unaltered. Any affine transformations or combination of transformations can be expressed in homogenous coordinates in terms of a matrix as shown in the equation below,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{m}_{00} & \mathrm{m}_{01} & \mathrm{m}_{02} \\ \mathrm{m}_{10} & \mathrm{m}_{11} & \mathrm{m}_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{4.1}$$

where the m_{ij} in the matrix are constants for any given transformation. It can be shown mathematically that all combinations of rotation, translate and shearing are affine transformations.

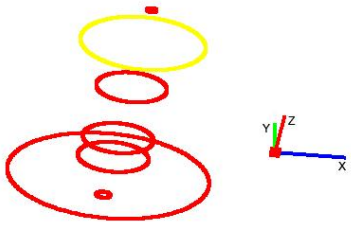


Figure 4. 5. The transformation axis

Each transformation updates the curves differently, for example scaling multiplies each axis by a given scaling factor that is calculated from the mouse movement,

$$\begin{aligned} x' &= s_x * x, \\ y' &= s_y * y, \\ z' &= s_z * z, \end{aligned} \tag{4.2}$$

where s_x, s_y and s_z are the parametric values that update the axes by multiplying with the original x, y and z coordinates respectively to produce then new scaled curve. In the interest of rotating the curve around a specified axis, first the axis of rotation is selected for the transformation to take place and apply the formula with a given angle of rotation. For example, Equation 4.3, states the new coordinates of any given point of the curve after rotating it around the x axis by the angle e,

$$\begin{aligned} x' &= x \\ y' &= \cos(e)*y - \sin(e)*z \\ z' &= \sin(e)*y + \cos(e)*z, \end{aligned} \tag{4.3}$$

the translation of a curve is obtained by simply adding or subtract a point to any given point. The new coordinates of any given point on the curve are now given by Equation 4.4,

$$\begin{aligned} x' &= x + t_x, \\ y' &= y + t_y, \\ z' &= z + t_z, \end{aligned} \tag{4.4}$$

where t_x, t_y and t_z represent the components of the translation point.

These transformations are applied locally on the curves. In each case, the centroid of each curve needs to be calculated and used according to the

selected transformation .i.e. for scaling, the centroid need to be subtracted from each new point .

The centroid C is computed using Equation 4.5,

$$C = \frac{P_1 + P_2 + ... + P_n}{n}, \tag{4.5}$$

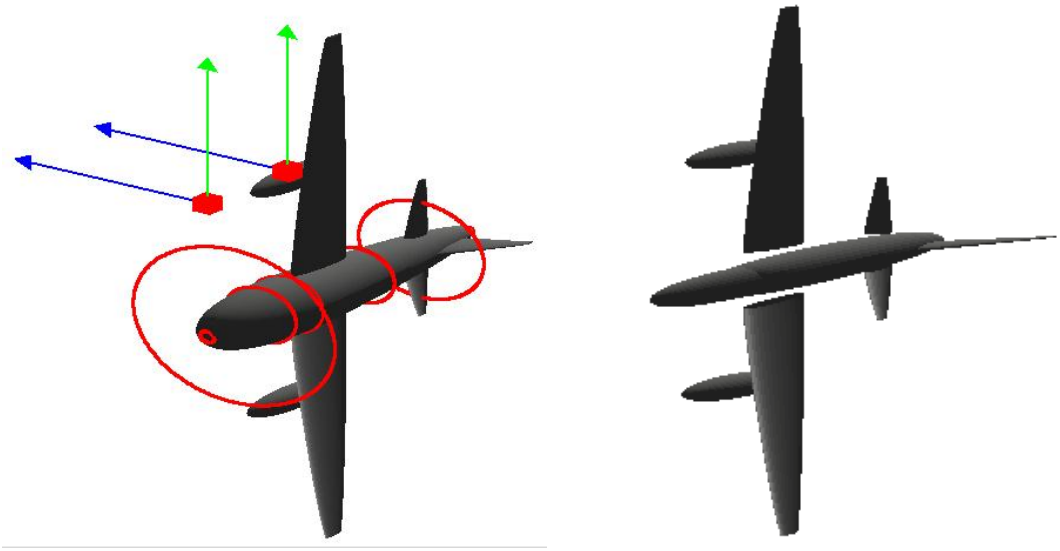
where P_n is the total points the curves contains in the x, y, z and n is the total number of points.

4.2.5 Parametric manipulation of the aircraft geometry

The curve generation process discussed in the previous section is responsible for dividing the aircraft design process into several designing steps, as well as including various tools for creating and manipulating the boundary curves. This section is focused on the manipulation of an existing aircraft configuration. One of the main advantages of using the PDE method in this work is that all the information required for calculating a surface can be found in the boundary data. The boundary data can be expressed as a set of curves that hold information about the shape of that surface. Once these curves are identified and extracted, the system will store them for representation and possible manipulation. The system includes a set of pre-configured curves representing various models that can be loaded and visualised from the application's main menu. Once a configuration is loaded, each part of the aircraft geometry can be individually selected and re-adjusted to generate new shapes or different configurations of aircrafts. The user can interactively select a transformation mode from the menu, such as translate or scale, and a directional axis for that transformation to take place, Figure 4.6. Each selected aircraft part contains a transformation axis different to the axis used before for designing the curves.

Rotation has been disabled since applying such transformation will deform the overall design. The new updated position will be then applied to each of the curves that are associated with the selected part; thus recalculating the PDE surface using the new updated boundary conditions.

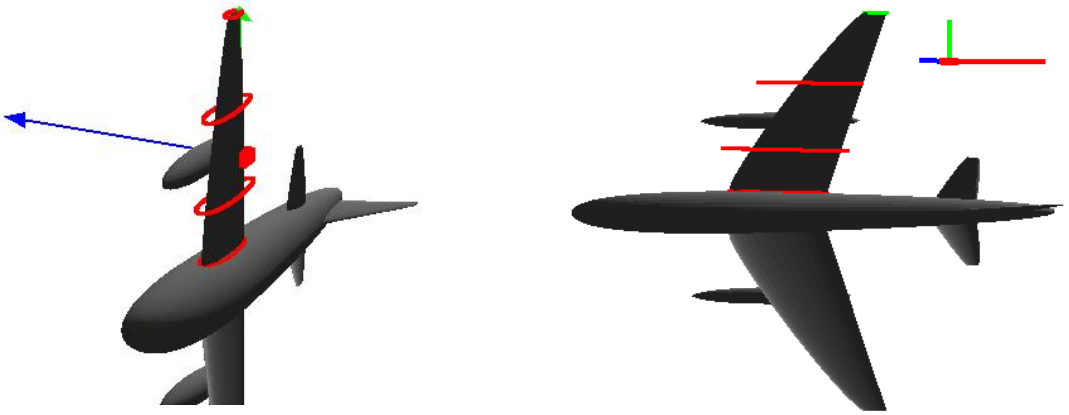
Figure 4.6 (a) shows the transformation vectors for scaling locally the fuselage component of the aircraft geometry. In this example, only the x and y coordinates are enabled to prevent deformations in the aircraft design process. The directional vectors will ensure that the transformation will take place only for the selected axis. The shape of some of the curves is seen in Figure 4.6. They appear to be much bigger than the actual surface representation. For the sake of guaranteeing a smooth surface, these curves have been used as the derivative conditions for the solution of the PDE method. Figure 4.6 (b), contains the resulting PDE surface for the fuselage after the scaling process. In this example the wing and rear tail wing components are detached from the main body since the transformation is applied only in the fuselage part of the aircraft. Blending any detached surface with the main body follows the same steps as discussed in the wing design section. This process requires projecting the starting curve of that surface to the (u,v) parametric space of the fuselage geometry. However, the required boundary data for the construction of that surface already exists in the system, thus making the process of blending two surfaces a very simple operation. The user will have to select the wing part and with the use of the keyboard directional keys, the initial curve will be projected as well as re-positioned on the fuselage in (u,v) parametric space, ensuring surface continuity.



(a) (b)

Figure 4. 6. Two vectors for controlling the fuselage shape, each vector transforms the associated surface patch (a). The resulting fuselage after the scaling, the wing and tail geometry needs to be projected to the fuselage to ensure surface continuity (b).

The process of manipulating the wing geometry is subject to some restrictions; only one transformation vector is available for global manipulation of this particular curve-set (Figure 4.7 (a)). This vector, which acts as a handle, translates or scales the selected curve-set in that particular direction; while the new generated surface will be automatically blended with the fuselage. Once the manipulation of the wing part has achieved the desired shape, the user can reflect the surface on the opposite side of the fuselage and therefore finalize the construction. Additionally, each curve of the selected aircraft component can be modified individually. Figure 4.7 (b), shows the local translation of the last curve of the wing configuration. The transformation process for each mode is different. For example, local manipulation of the starting curve of the shape is not possible, since that curve needs to be projected onto the fuselage surface and re-position the rest of the curves for that surface. This process can be also applied to manipulate the horizontal and vertical rear tail wing component according to the aircraft configuration.



(a) (b)

Figure 4. 7. Transforming the wing geometry.
Global manipulation of the wing configuration (a).
Local manipulation of the wing configuration (b).

This technique gives us the advantage of modifying the overall shape of an existing aircraft design by simply applying a series of transformations to the boundary curves. The aircraft geometry can be re-evaluated each time the boundary curves are adjusted. The use of the PDE method, results in a surface that can be re-computed with different resolutions only by adjusting the resolution of the u,v parametric space the surface lies in. The advantage of this operation is that it minimizes the storage requirements while maintaining the quality of the output surface, since only a small set of control points is required to reproduce that surface. Next section contains various PDE-based aircraft configurations constructed using the PDE aircraft designer system.

4.3 Examples

4.3.1 Delta wing configuration

Figure 4.8, consists of a delta wing configuration airplane. This configuration requires following the steps explained in the previous sections. The procedure is described in different steps in Figure 4.8. The first screenshot shows the construction of the wing curves on one side of the aircraft. As discussed before, the first curve of the set needs to be projected onto the fuselage surface to ensure surface continuity. This curve will be used as a guideline for the rest of the wing curves. Once all the new wing curves are inserted into the system, the shape of the curves can be modified by applying a series of transformations. The wing geometry will be computed and visualised using the PDE method. To complete the wing configuration, the resulting PDE surface must be reflected on the opposite side of the fuselage. This operation will generate a mirrored curve-set of the current wing configuration and generate a PDE surface for these new boundary data. Depending on the desired aircraft configuration additional steps might be required; Figure 4.8 (b) and (c) shows the construction of the horizontal and vertical tail geometry by applying the same technique of local and global transformation of the boundary conditions. Finally all the parts are constructed and rendered in the application as shown in Figure 4.8 (e).

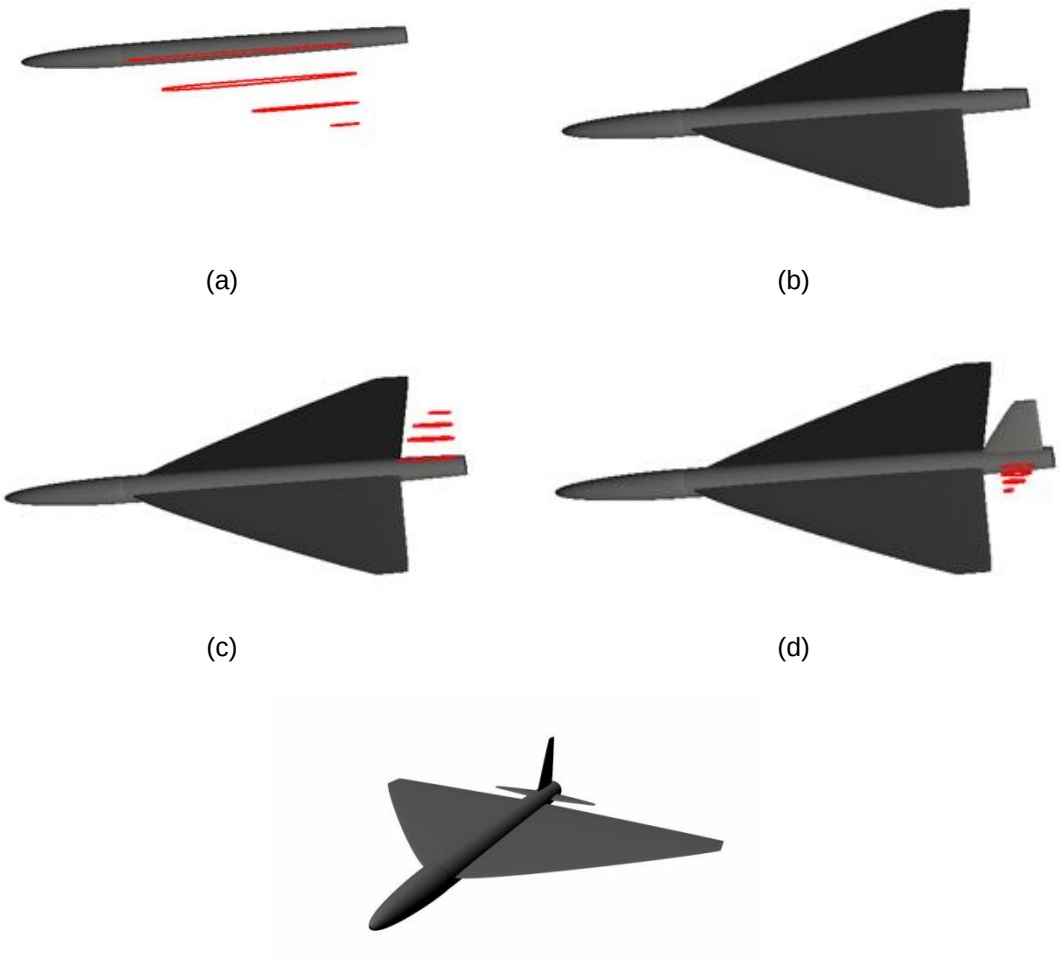


Figure 4. 8. Constructing a Delta wing aircraft configuration.
Creating the left wing part (a). Reflecting left wing to the right side (b).Creating side wing tail parts (c) (d). Final PDE-based surface representation of a delta wing configuration.

4.3.2 Double delta configuration

The second example shown in Figure 4.9 contains a double delta wing configuration. An alternative approach for constructing such configuration will require loading the previous delta wing configuration and manipulating the already existing geometry. The major difference in this design is the double delta shape of the wing components and in order to achieve such a representation, the wing curves need to be initialized with 7 curves. This extra information will produce a two-patch PDE surface, thus giving more flexibility

over the manipulation of the geometry. A series of global and local transformations need to be applied to the curve set so that the desired double delta shape to be achieved. Figure 4.10 show various configurations designed using the plane designer application. The engine component is an additional part that can be inserted from within the application. It consists of a set of four cylindrically shaped curves that can be transformed only globally. This part is used here only for visual purposes and it can be improved to include various pre-configured engine designs.

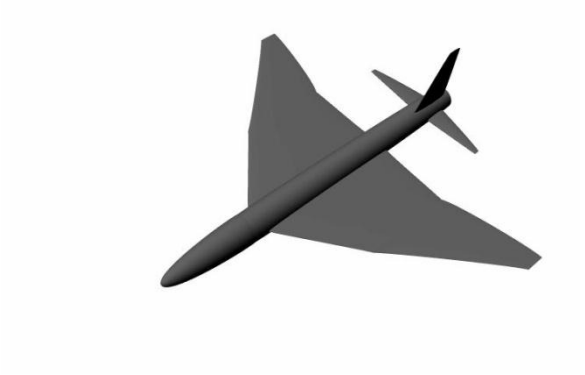


Figure 4. 9. Double delta wing configuration.

(a) (b)

(c) (d)

(e)

Figure 4. 10. Various PDE-based aircraft configurations.

4.4 PDE Maya plug-in

The next examples focus on the integration of the PDE method to different technologies. This application was originally developed by Eyad Elyan [61] as a Maya plug-in for constructing PDE surfaces. An expansion of later work has been carried out as part of this project; this included the manipulation of PDE-based polygon surfaces using the PDE method. Maya is a popular modelling and animation CAD package that contains various tools for the interactive design and manipulation of mesh surfaces. From an implementation point of view, the system requires a set of minimum four curves. More complex shapes will require higher number of curves and tools than will enable manipulation of control points. Availability of such tools within a visualization environment enables the user to create and control curves with more ease. The aim of this work is to utilize Maya's functionality in the creation and manipulation of PDE surfaces as well as enable the use of PDE surfaces from within the application.

The integration of the PDE method was achieved using the Maya Software development kit (SDK) [62] that contains a rich set of tools that makes possible adding new functionality to the software. The SDK includes a Maya C++ Application Programming Interface (API) that provides functionality for adding, querying and changing Maya objects to create optimized and robust tools [63].

The use of the API, adds new functionalities to Maya such as: commands, shaders, animation nodes, graphical manipulators, geometry shapes, dynamic fields, particle emitters and any type of custom node. Figure 4.11 below contains an example of PDE surface constructed in the Maya environment with the use of the Maya PDE Plug-in.

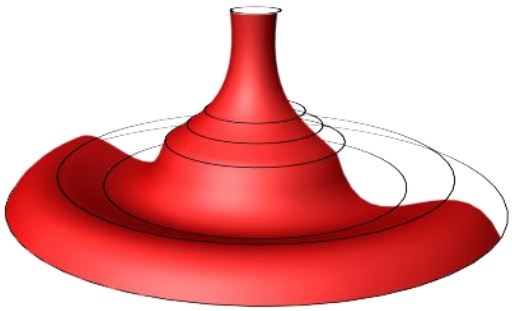


Figure 4. 11. PDE surface calculated between 0 to π domain.

4.4.1 Integration of the PDE method

The plug-in is utilizing the Maya node based system, which implements various commands for rendering, animation and modelling [64]. These commands are available from the C++ API and enable accessing the contents of a Scene graph, selection list and various other functions that can be used to communicate with the user's input and generate a PDE based geometry. Maya is a very efficient database for storing graphical information. All the information is stored in the Dependency Graph (DG) [62]. The Dependency Graph is a collection of entities or nodes connected together. Connections in the graph allow data to move from one node to another. In summary, the DG is the heart of Maya, while it performs the typical rendering, modelling and animation; it also handles the communication between multiple networks of nodes. All objects in a scene and their data connections are represented by nodes and connections in the dependency graph. Nodes have properties called attributes that store the

configurable characteristics of each node. Similar types of attributes can be connected together, letting data flow from one node to another. A node has a set of input and output parameters that define its characteristics. One node's output could be an input for another. Several nodes can connect to each other to inherit their attributes. This data flow between different nodes can be used as construction history where if only one of the inputs of a node in the Dependency Graph is changed, it affects all the connected nodes in the hierarchy. There are various types of nodes that can be programmed, some of them are:

- Geometry deformers, contain various tools to enable surface manipulation. These include linear and non-linear deformers such as lattice, wrap and twist, bend, wave, sine etc.
- Dynamics emitters. These enable dynamic simulation by utilizing particles to create various effects such as waves, fire etc.
- Shaders (including hardware shaders). Various shaders nodes can be connected to each other for creating surface materials such as Anisotropic, Blinn, Lambert, Phong etc.
- Shapes. Includes various pre-build shapes such as sphere, cube, pyramid, cylinder, torus etc.

Taking advantage of the Maya Node system [63], the application generates a PDENode that takes as an input parameter a set of minimum four curves in the 3D space and outputs a PDE surface object. Once the PDE node receives the input curves from the selection list of the DG, it will calculate the PDE method for the given boundary conditions and visualise the resulting geometry in the Maya viewport. Every change in the boundary curves will trigger the PDE node

to re-calculate and display the new updated PDE surface in real time. Additionally, a new panel will be added in the Attribute Editor containing parameters for adjusting the PDE method. The user can manually update the u and v coordinates in the parametric domain of the output geometry, resulting in smoothing the surface. The editor panel can also control some internal parameters that affect the PDE method. Figure 4.12 shows a PDE cylinder evaluated in different resolution levels created from adjusting the u,v parameter domain. Other parameters include updating the number of control points per curve, specify the range v between of 0 – 2π and changing the mode of the PDE solution. Note that the mode of the PDE solution can be adjusted from the graphical interface in the Maya environment and can control the approximation of the resulting surface to the boundary conditions.

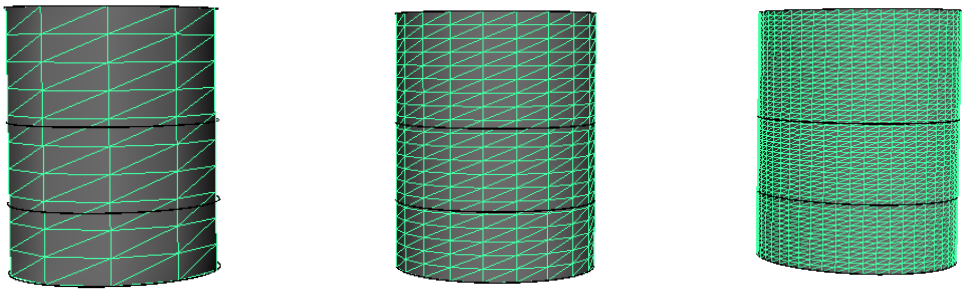


Figure 4. 12. Cylindrical PDE-based surface obtained though the Maya plug-in developed in this work. Surface resolution 10x10.(a) The resulting PDE surface using a resolution equal to 20x20 resolution.(b) 50x50 resolution.(c)

The Figure 4.13 (a) below shows the initial boundary conditions required to compute a two-patch PDE surface, shown in Figure 4.13 (b). Additionally, the user can control the boundary curve in order to further manipulate the object. Simple affine transformations such as rotation or scaling can be applied directly to the curves to control the shape. The PDE method is then re-calculated every time the input parameters are updated. The output node can be connected with

a custom rendering node to control the appearance. Maya provides various rendering nodes that can enhance the quality of the rendering, as well as create various effects such as wood, metal, plastic, water, etc which can be used directly onto the PDE surface.

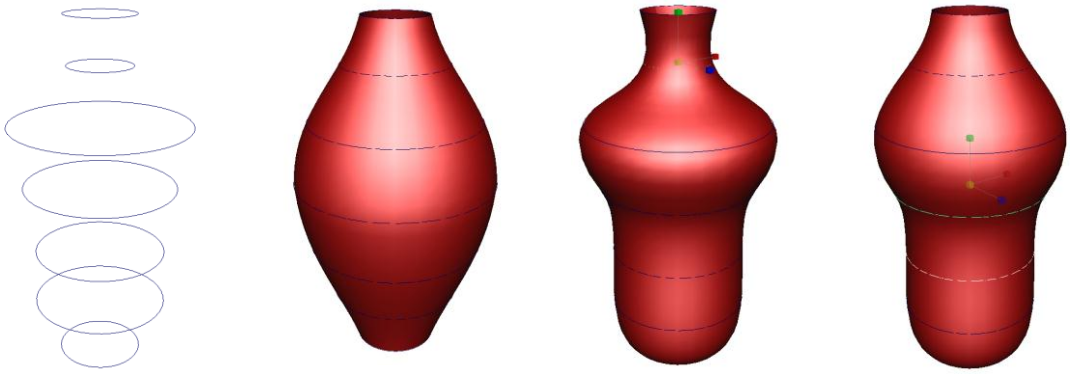


Figure 4. 13. Manipulating PDE geometry.

An example of a more complex geometry is presented below in Figure 4.14. Here the curves showed in Figure 4.14 (b) represent a region of a 3D scan mesh shown in Figure 4.14 (a). The face has been segmented into meaningful regions such as eyes, nose, mouth etc. Each region is controlled by a set of curves; a total 28 curves generate 9 PDE surface patches. Figure 4.14 shows three different resolutions of the same object. The user can choose between several versions of the same object depending on the end-user need. Additionally, only a small set of control points is required to calculate a surface representation. The Figures 4.14 (c) (d) (e) have been calculated using a resolution of 10 x 10, 20 x 20 and 50 x 50 in the discretized u, v resolution accordingly.

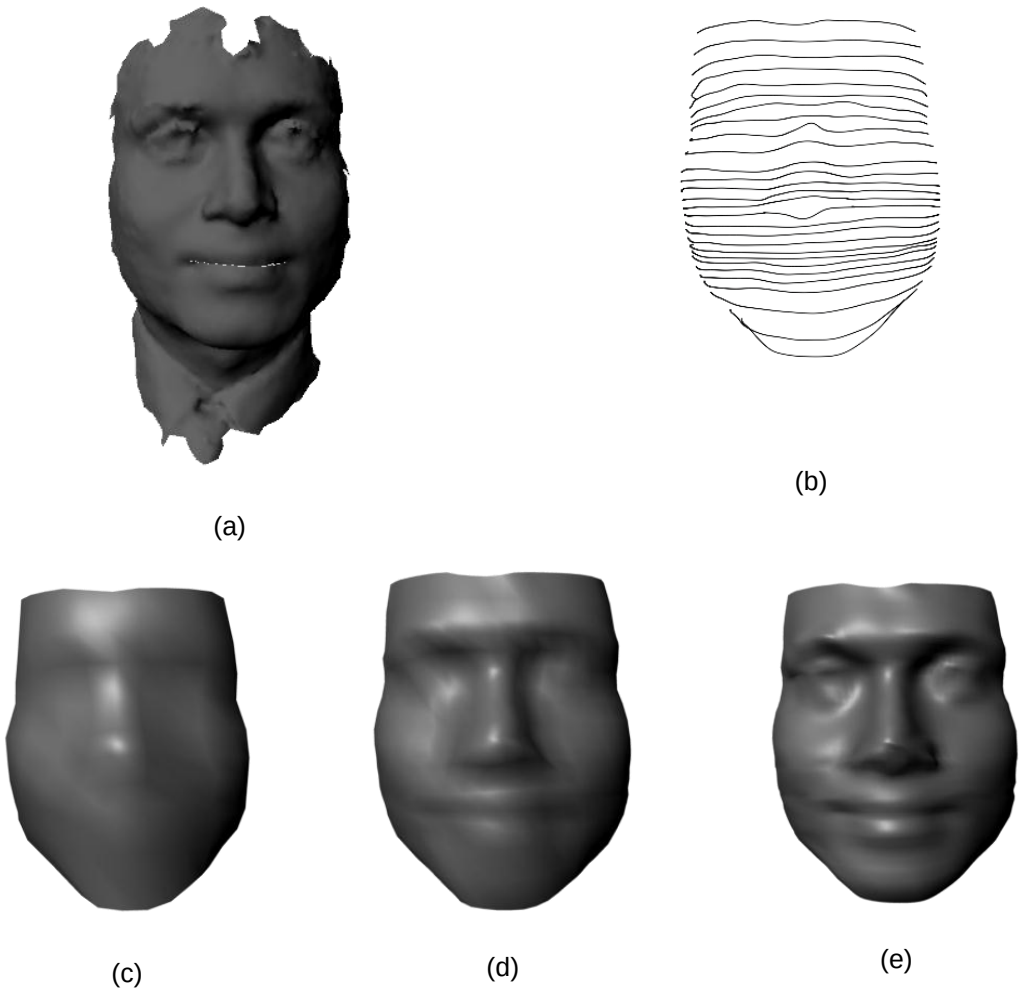


Figure 4. 14. The original scan model of a face (a). The boundary curves that represent the face geometry (b). The PDE-based surface representation of the scanned face evaluated in three different resolutions (10x10, 20x20 and 50x50 respectively) (a, b, c).

4.4.2 Global manipulation using the PDE Maya plug-in

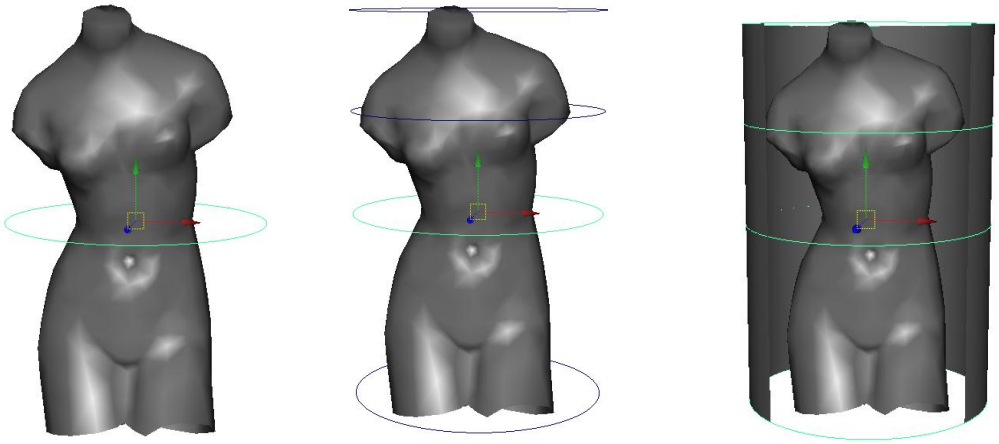
As mentioned before the PDE Maya plug-in was developed by Eyad Elyan [61] as a toolset for constructing PDE surfaces in Maya. His work was expanded to include global PDE and mesh surface deformation capabilities [65]. A new technique has been introduced to manipulate both PDE and polygon surfaces using the PDE method; this technique can be used as a modelling tool to manipulate globally the geometry of complex surfaces.

This new feature requires a given polygon surface or a set of curves and a constraint curve positioned where the global manipulation is required. The

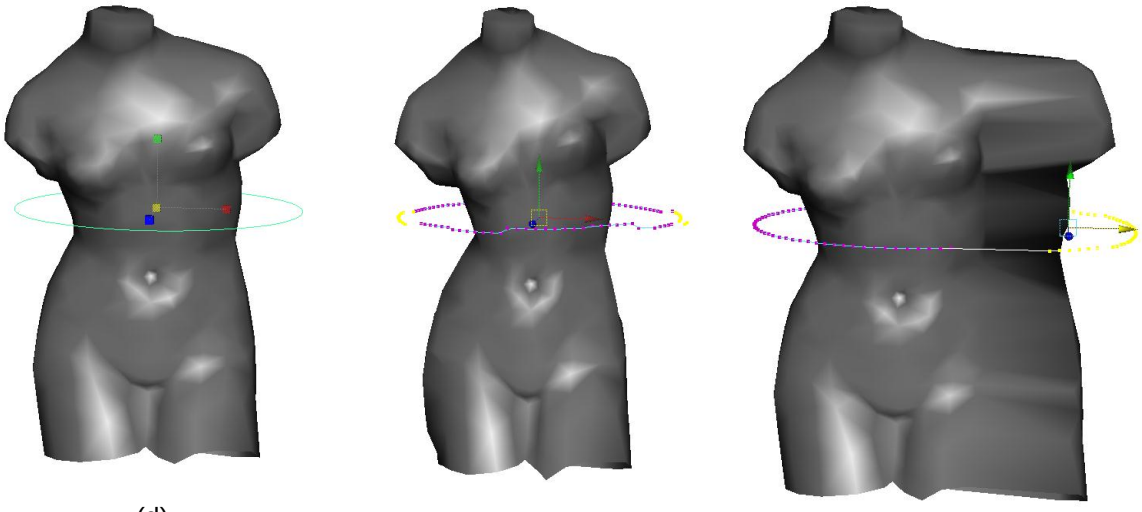
process consists of generating a set of curves around a given mesh by using the constraint curve as a guideline. The user is able to design a curve around the area that need to be modified, this curve will be used as the constrain curve and as a handle for the manipulation of that surface. The constraint curve is duplicated to generate three additional curves; the bounding box of the mesh is computed to identify the minimum and maximum points of the object. Each of these new curves will be then positioned inside the bounding box; the order is: First and last curve will be placed in the maximum and minimum position of the bounding box accordingly. The second curve is the constraint curve and third curve will be the difference of the first and second curve, Figure 4.15 (b). Next, the PDE surface is computed, Figure 4.15 (c). The resulting surface has a cylindrical shape that contains the input mesh object. Note that the local PDE surface and the control curve set contained in Figure 4.15 (top), are not visible to the user and they are only used internally for manipulating the input geometry.

The constraint curve is internally linked to the computation and transformation of the new cylindrical PDE surface and it will dynamically reconstruct and manipulate the object every time its control points are adjusted [66]. It is used as a handle to control the transformations of the geometry. The global manipulation of the mesh is achieved by associating each point of the mesh model with the nearest point of the PDE surface; and finding the difference between the PDE surface and the target mesh for each point. Manipulation is carried out by adding that difference to each point of the original mesh model according to the mapping correspondence previously found. This ensures that the position of all the translated points of the PDE surface will be transferred to the corresponding on any given mesh.

Figure 4.15 (bottom), shows various examples of surface manipulation. The user can directly manipulate the control curve or a set of control points to achieve the desired shape. Additionally, the number of control points in the constraint curve can be increased to improve the results of the manipulation. That will generate more points on the PDE surface, which in return produces a better association and a smoother deformation of points between the two surfaces. Various properties that define the PDE surface can also be adjusted from the PDE panel in the Maya environment. The resulting mesh can be re-processed again by repeating the procedure as many times as needed. The choice of a different constraint curve results in a different PDE configuration and can enhance the overall manipulation.



(a) (b) (c)



(d) (e) (f)

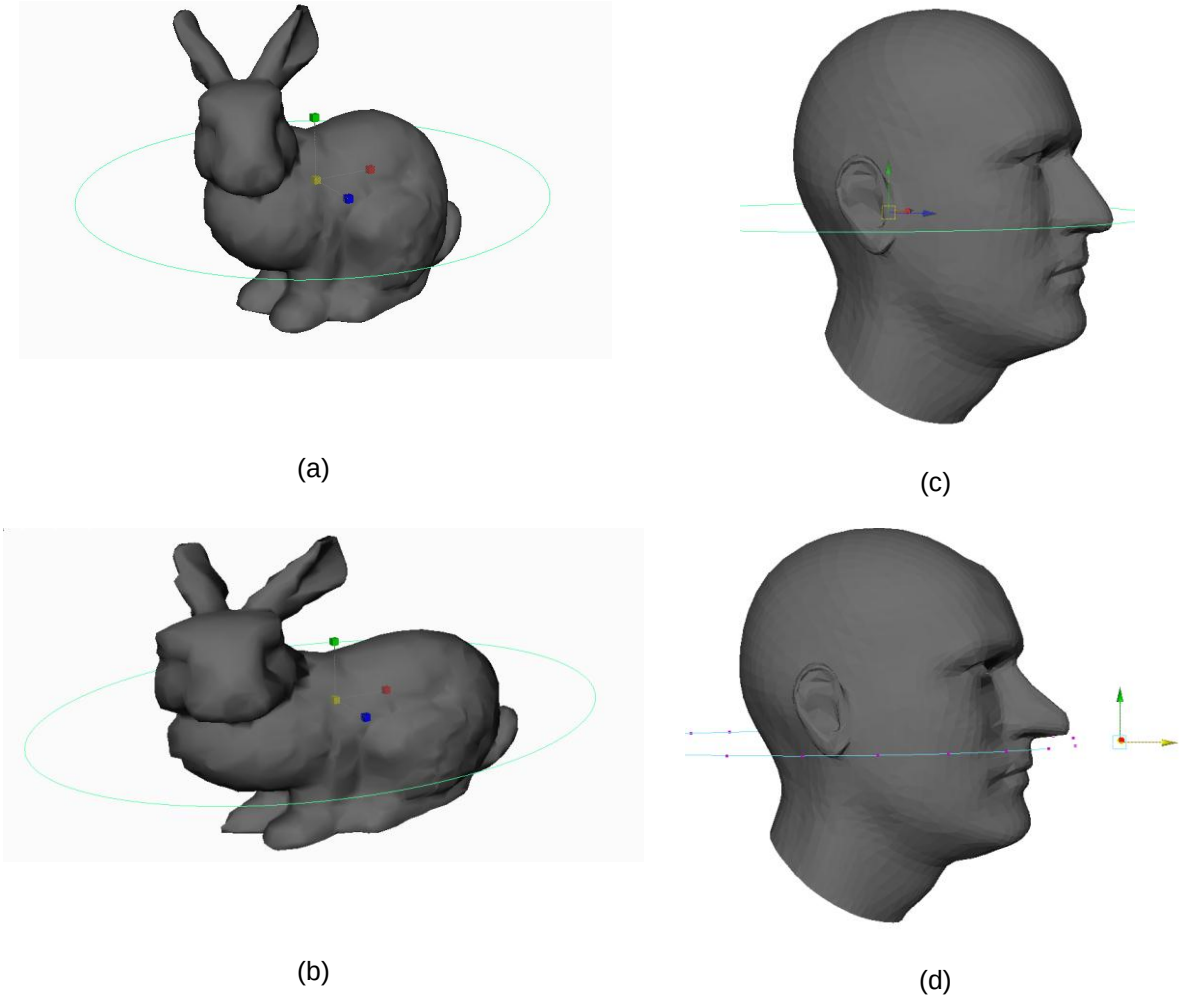
Figure 4. 15. PDE-based global deformations of a mesh surface.
Initial mesh surface and constraint curve (a). Boundary curves representing the local PDE-based surface (b). The local PDE-based surface used for the deformation (c).Examples of global deformations (d, e, f).

Although the technique has been applied directly to polygon surfaces, it can be used for PDE surface as well. The process follows the same steps, with the exception of using a set of curves instead of a mesh for input geometry. The input curve set is used to calculate the PDE surface that represents the mesh for manipulation. The procedure is afterwards the same: another set of curves is generated and used to calculate a local PDE surface that includes the mesh for manipulation. And lastly the association of every point of the two geometries is used to manipulate the surface.

4.5 Examples

Examples of PDE based global manipulation are shown in Figure 4.16. The objects have been deformed using the PDE Maya plug-in; the face in Figure 4.16 (c) (d) shows the global transformation of the nose area. A set a control points are selected and translated in the x axis direction, the deformation

occurs in real time and it is controlled by an inner PDE surface that is recalculated every time the curve is adjusted by the user.



(a) (c)

(b) (d)

Figure 4. 16. Original Stanford bunny model (a). PDE-based surface deformation of the Stanford bunny model using a global scaling (b). Original human face model (c). Deformations of a human face model by selecting control points on the constraint curve(d)

4.6 Conclusions

The surface generation software presented in this chapter has been designed for the construction of generic aircraft shapes. The surface representation is based on the PDE method and is fully parameterized so that completely different aircraft configurations can be generated. Unlike other techniques, the PDE method can parameterize complex surfaces in terms of a small set of design variables, instead of hundreds of control points. The shape of the surface is then defined through boundary conditions and a small set of design parameters. Taking a boundary value approach to the problem of surface design has the advantage that most of the information required to define a surface is contained at the boundary or character lines representing that object. The representation and the control of aircraft shapes is achieved through the adjustments of parameters associated with the curves. Thus, the user can interactively transform the parameters and change the shape of the aircraft without having any knowledge of the mathematical theory behind the PDE method.

The software is designed in a user friendly interactive manner in order to allow the user to design and manipulate any aircraft configuration in real time. Future improvements may include: additional aircraft parts, various tools for designing accurate airplanes such as the addition of materials, colours and textures to the airplane surface to make it look more realistic while exporting 3D objects in various formats so they can be used within commercial CAD packages. Future work could also be undertaken in either aircraft simulation by using a dynamics engine or a stress/strain analysis; for instance, simulating an aircraft under

severe weather conditions or applying stress analysis to determine the stress in materials and structures subjected to static or dynamic forces.

The aim of this second technique presented here is to expand the PDE method capabilities to enable surface deformation and offer an easy way of constructing and transforming PDE surfaces within a 3D environment. This has been achieved by providing a graphical and user friendly toolset within the Maya environment to generate and manipulate complex geometry in real time. The PDE surfaces can be either deformed locally, by adjusting the control points of the boundary curves that is associated with; or globally by using an internal PDE surface, controlled by a constraint curve to calculate the difference between these two surfaces and add it to the original mesh. The technique is still under development and there is still room for improvement. Future work includes a better inner PDE configuration to improve the association of the two geometries and the overall deformation as well. Additionally, interactive selection of various areas can be integrated to enable direct local based deformations of a given mesh.

Chapter 5: On the development of PDE-based animation

5.1 Introduction

Animation of a human character is a crucial problem in computer graphics. Depending of the end-user application, the geometric model resembling a human is subject to different constraints. For instance, one of the major tasks within the video game industry consists of minimizing the computational cost while keeping the motion as realistic as possible. To that extend, there are various constraints to take into account when modelling a motion such as external forces or adapting the gait to the environment. In other fields of computer graphics such as video and films, the requirements are realistic animation of detailed and accurate models without caring about the computation cost since the rendering takes place in real time. Nowadays, several commercial packages provide the user with tools that automatically generate human-based animation and can manage motion-capture data. Animating complex models such as humans is usually done by creating a simple representation of the model, called skeleton. The motion is first computed for the skeleton, which can be displayed and manipulated interactively within a CAD or a visualization environment. Once the animation reaches satisfactory levels, it can be applied to a human character model as a "skin"; the surface representing the character model. Some of the requirements that must be taken into account before modelling a cyclic motion are:

- The character animation must look realistic.
- Animation must react and adapt to the surrounding environment. (Collisions, terrain slope. etc).
- Animation must allow changes in the motion of the character. (Lifting weight, external forces, etc.).

These requirements can be less or more important, depending on the end-user application. In virtual environments for example, environment adaptation and interaction is more important than computing realistic forces and torque. The aim of the chapter is to examine various techniques for computer graphics animation, as well as introducing new techniques for generating PDE-based cyclic animation.

5.1.1 Key-frame systems

Key-frame was originally developed by Walt Disney to generate traditional animation [67]. In such systems, animators would design a particular sequence by drawing frames that establish the animation. The realism and the quality of the animation were depending entirely on their skill to animate realistically a character they had designed. This technique was later implemented in computer systems, where the animator could interactively create and adjust key-frames as a sequence of loops. In computer animation the term key-frame applies to any variable whose value is set at specific key-frame and its values for the intermediate frames are interpolated according to some procedure. These systems usually provide an interactive interface from where the animator can specify the key values and the desired interpolation technique [68]. A common procedure to generate animations consists of generating different series of key-frames and using various interpolation techniques to approximate the in between positions. Various methods have been developed to automatically generate series of key-frames based on the basic human motion mechanisms, thus providing parameterization of the resulting motion [69]. The motion can be controlled by parameters such as step length and step frequency. A key-posture is associated with each key. These postures are linearly interpolated to produce the in-between angular values. While

interpolation of key-frames remains a fundamental technique for most animation systems, as a standalone technique for animating contains various disadvantages. Some of them are:

- It is only suitable for simple motion of rigid bodies.
- Special care must be taken to ensure that no distortions on the surface are introduced.

As the need for more complex animation increased, new techniques were introduced to automate the generation of human-based animation. Some of them will be presented in the following sections.

5.1.2 Articulate figure animation

Articulated figure animation which is based on Forward/Inverse kinematics, became popular since it relies on the understanding of the basic human motion mechanisms [70]. An articulated figure also called “skeleton” is a hierarchical structure that consists of a series of rigid links connected at joints as shown in Figure 5.1. Thus, the quality of the motion is based on the quality of the model representation or skeleton rather than on the artist’s skills. Most of the theory on kinematic animation in computer graphics comes directly from the field of robotics. Robotics is concerned with all types of joints in which two links move relative to one another; whilst the computer graphics approach is mainly based on revolute joints (one link rotates about a fixed point of the other link). These links are usually considered to be pinned together and the link further down the hierarchical chain rotates while the other one remains fixed.

Structures with more than one degree of freedom are called complex joints. These include planar and ball-and-socket joints. Planar joints are the ones with one link slides on the planar surface of another. Whereas, a ball-and-socket

joint is capable of motion around any number of axis, enabling it to rotate 360 degrees. Usually, when a joint has more than one degrees of freedom (DOF), such as ball-and-socket joints, it is modelled as a set of n one degree of freedom joints connected by $n-1$ links.

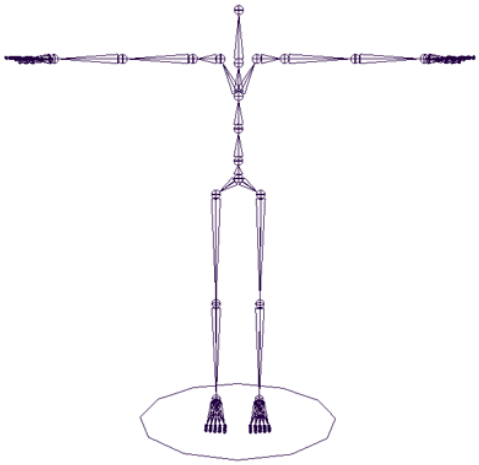


Figure 5. 1. Skeleton system

The representation of a hierarchical model is usually done by a tree structure of nodes linked together by arcs. In this representation, the highest node of the tree is called the root node. The position of all the other nodes of the hierarchy is located relative to the root node. A node that is higher up in the hierarchy than another node is called a parent node and the one below is called the child node. In the tree structure there is always a root arc that represent a global transformation to apply to the root node and to all of the nodes in the tree. Changing this transformation will re-position the entire structure. The animation of articulated structures in human animation usually falls into one of the two following categories: Forward and inverse kinematics.

5.1.3 Forward kinematics

In forward kinematics the motion of all joints is specified by the animator. The motion of the end effectors (hands and feet) can be determined indirectly as the accumulation of all the transformations that lead to that end effector [70]. In a case of animating a figure's foot, the motion is computed from the combined effect of the transformations at the hip, knee and ankle. Equation 5.1 below shows the function for calculating the new position x for a given angle,

$$X = f(\theta), \tag{5.1}$$

where x derives from given angle θ.

In this formulation, the hierarchy will be evaluated by traversing the tree according to the setting of the joint's parameters from the root to the child nodes. The traversal will search the tree until an unprocessed downward arc is found. If the arc is found, it is processed and the traversal continues to the next unprocessed arc. The procedure is continued until all nodes and arcs have been visited.

The main disadvantage of using forward kinematics exclusively is that it requires manual design of the poses that need to be processed. In the case of a human model the animator will have many transformations to control for each body part, which may prove to be too complicated. The procedure of getting the model to the final desired position by specifying joint angles can be a very difficult task for the user; a task which often is a trial and error process. Additionally, the use of forward kinematics makes difficult to add constraints to the motion, for example specifying that the feet should not penetrate the ground

during a walk cycle. In many cases, available biomechanical data can help to automate the generation of key frames, while interpolation techniques are used to generate the in-between positions.

5.1.4 Inverse kinematics

In this technique the animator defines the position of the end effectors only. It is used in applications where the movement of the end effector drives the animation of the hierarchy. Such an animation includes everyday motions such as running, walking or arm positioning. Attempts to animate such movements using forward kinematics can prove difficult. The procedure of generating the animation by inverse kinematics consists of placing the figure in a sequence of desired poses closely spaced in time and space, giving the illusion of motion. For each of these given poses, the joint angles that are required to maintain that configuration are calculated analytically. Given an initial pose vector and the final pose vector, intermediate configurations can be formed by interpolating the values of the pose vectors. Equation 5.2 below shows the function for calculating the angle that is required to move the joints by specifying the position,

$$\theta = f^{-1}(X), \tag{5.2}$$

where angle θ is found from the given position X .

Inverse kinematics is considered a hierarchical network of nodes or skeletal structure. When travelling down the hierarchy from the root, every node is accessible, while moving up the hierarchy from any node always leads to the root node. Inverse kinematics can be applied between any nodes in the

skeleton and can specify the position and orientation of all nodes between the end node and the base node. Note that the node corresponding to the end effector, also called the end node, is lower down the hierarchy than the node corresponding to the base node. This can be seen as a constraint which inverse kinematics tries to satisfy. This constraint can be used independently between the skeleton system and the virtual environment by keeping the joint limits correctly constrained while carrying out a particular action.

Most of the kinematic approaches used for generating human-based animation rely on a combination of forward and inverse kinematics for computing the motion. One of the main advantages in these kinematic models is the high level of parameterization they provide. Attributes such as position, velocity and acceleration can be included in the computation, leading to the generation of different gaits. Additionally, they can offer low computational cost, for example the complexity of inverse kinematic algorithms is $O(n^3)$ [71] depending on the number of joints. Obviously the inverse/forward kinematics approach is not the only factor involved in animating of articulate structures. The motion of rigid joints is not always suitable for natural articulated structures. In the case of vertebrate mammals, the motion occurs from the flexing of the spine, which is not a rigid joint. Different approaches exist to generate animation that can guarantee realism will be discussed in later sections.

5.1.5 Dynamics animation

Dynamics animation can add various constraints to a predefined motion or can directly generate a cyclic motion such as a walk or a run. The simplest form of dynamics animation is where a mathematical model is used to manipulate the geometry of a surface as well as its movement. In these models various dynamics theories are used to calculate surface deformation or collision behaviours with another object. Fundamental laws of dynamics are applied to an articulate hierarchical structure to improve the realism of the motion. Various examples that are using this technique include flocking animation [72], crowd animation [73] , cloth animation [74], simulating water animation [75] and fish locomotion [76]. In the case of fish locomotion, the motion can be calculated as a sinusoidal function with different levels of amplitude. In this particular case, an analytical model can be used to simulate the forces that produce the motion.

An example of fish animation can be found in [76], where factors such as social behaviour and collision with external objects have been included in the calculation of the motion. This work uses a mesh representation of a generic fish and adopts a spring mass model that calculates the natural movement of the fish as well as the collision response. The spring model consists of four springs called motor controllers that control the amplitude and the speed of the movement. Generally, spring-mass models are solved using numerical techniques such as the semi-implicit Euler method [77, 78]. Such numerical methods can be either slow or prone to numerical errors, which may lead to inaccuracies affecting the geometric model. One of the main advantages of using such approaches is that the motion is produced from visually observed and controlled data which leads to a more realistic calculation of the motion.

5.1.6 Cyclic animation

Understanding the interaction of various joints involved in the process of locomotion is a crucial step to produce realistic human based motion. Various every day actions such as walking, running or reaching for an object can be expressed as a form of a cyclic movement [79]. Walking, as an action, is responsible for translating the model from one position to another and at the same time maintaining balance and different gait characteristics. However, in this case, dynamics plays a much more important role in the formation of walking motion than it does in reaching for an object. Knowledge of the walking motion can be expressed as a set of biomechanical data or a set of adjustable parameters that can be used as a global control mechanism for generating a distinct gait. Attributes such as stride length, hip rotation and foot position can be used to specify a particular pose. Additionally, kinematics is used to create constraints on various joints rotations such as movement duration and positioning, while dynamics forces and torques can be then added to the defined pose to satisfy these constraints and improve the physical movement.

A cyclic animation is defined as a loopable animation of a character repeating an action, and can be used in a hierarchical structured animation to move a character along a path easily and realistically. The repetition of a phase sequence is called a cycle and the time taken to complete is referred as the period. Furthermore, a phase within a cycle can then be defined from a set of parameters responsible for a particular movement. For example, a duration leg factor can be used to describe the time elapsed between the moment the leg is on the ground and when it generates a distinguished locomotion. By animating a single cycle, rather than manually animating every footstep, the animator can save time. This technique is ideal for repetitive motions, such as a character

walking, running, swimming, etc. For example, in the case of walking, the character is animated taking a step with its right foot and then a step with its left foot. The loop is created so that when the sequence is repeated, the resulting motion is seamless.

After examining the human walk in more detail (Figure 5.2), it can be characterized as a sequence of phases separated by foot strikes and takeoffs. A foot strike refers to the moment when the foot is in contact with the ground while a takeoff describes the moment when the foot leaves the ground. In gait terminology, a complete cycle from one takeoff to another takeoff is defined as a stride, while the cycle between the takeoffs of the two feet is called a step. Four foot strikes and takeoff events occur during a stride. A run cycle can also be described as a sequence of phases. The main difference from a walk cycle is that there are phases where both feet are off the ground.

Figure 5. 2.Sequence of phases for a walk cycle.

5.1.7 Spine animation

Another approach for generating realistic animation is by utilizing the spine of a PDE-based surface. One of the many advantages of the PDE surfaces is the ability to create and manipulate complex shapes with ease. In CAD environments the user has little or no knowledge about the mathematics behind the calculation of a PDE surface and the manipulation of its boundary data. Thus, it is crucial to adapt a technique that allows surface definition as well as manipulation. The spine of a surface is an entity that characterizes the object's topology and describes the object by a lower dimension [29]. It has a very close geometric resemblance to the object's medial axis or topological skeleton. It is not widely used in CAD systems due to the lack of implementations that support spine generation procedures since the introduction of this types of surfaces is fairly recent. The spine of a surface can be represented as a cubic polynomial that can be used as a shape manipulation tool to deform the shape in an interactive manner. Due to the analytic form of the solution used to generate the PDE surface, the spine of the surface can be computed as a by-product of the solution [80]. This outlines the advantage of using PDE surfaces since most of the parametric surface generation techniques have to calculate the spine separately.

Utilizing the spine of a surface for producing animations can be classified as dynamics animation since it can produce and manipulate a motion by means of mathematical functions. Mathematical functions can be applied to the spine to generate oscillating effects on the surface. The surface can be then controlled by analytic expressions that will adjust the movement according to the input parameters. One of the main advantages of using PDE based spine animation is that a mathematical model is used to specify the geometry of a model as well

as its movement. Additionally, the spine can be animated in a cyclic manner since the general mathematical nature of a motion can be expressed using a travelling wave advancing in the opposite direction of the motion. These types of wave like motions are more suitable for generating animation of vertebrate mammals, since the motion occurs from the flexing of the spine, which is not a rigid joint.

In the case of the fish locomotion, the motion can be calculated as a sinusoidal function with different levels of amplitude. In this particular case, an analytical model is used to simulate the forces that produce the motion. More details on parameterizing fish locomotion using the PDE spine can be found in Section 5.4. In this example the fish movement is divided into four main categories. A mathematical function is then used to obtain the spine of the surface at different positions and parameterize the fish movement. Additionally, the motion of the fish is transferred from the PDE surface representation to an enhanced mesh model that representing the fish.

5.2 Skinning

Once the motion of the articulated figure has been defined, the next step is to attach it to a “skin” surface that will be controlled by the skeletal structure. This process is called skinning and is a mesh deformation technique that can be applied to soft body surfaces such as character figures and animals, as well as rigid surfaces such as robots. There are two widely used approaches for skinning a human character; surface and layered model. The surface model consists of a single outer skin surface, usually the human body model, and the skeleton structure that contains the animation. The human body model is

usually represented from either polygons or parametric patches such as NURBS or PDE surfaces.

The most common surface skinning technique is linear blend skinning; which is a mesh deformation technique implemented in most modern 3D engines, most frequently used for characters driven by skeletal animation in computer games. In this case, only one input skin mesh is provided. The skeleton-to-skin binding is defined in a geometrical way. Every vertex of the skin surface is associated with one or more skeleton joints. When a skeleton joint is rotated, the appropriate vertices are deformed to simulate the motion. Skinning a single surface is a fast method but it can generate several distortions on the surface if not applied accurately.

Better results can be obtained with additional computation cost; weighting the vertices surrounding a joint can guarantee that their position is affected by multiple bones. While weighting the vertices can result in smoother skin deformations around the joints, severe distortions can still occur depending on the skeleton's design. The placement of joints in the body affects the realism of the surrounding deformation. To avoid such distortions, joints need to be placed according to anthropometric data [81]. Additionally, replacing the linear blending technique with a more sophisticated blending technique can improve the skinning quality.

Another common approach used for human skin deformation is the multi-layered structure model. This technique was introduced by Chadwick, Haumann and Parent [82], their approach was to include an articulated skeleton, the surface geometry representing the skin and an intermediate muscle layer, which connects the other two together. The layered structure consists of three layers:

a skeleton layer, a muscle layer and a skin layer. The skeleton layer represents the skeletal structure of the human body, which consists of joints and bones. The muscle layer represents the muscular structure of the human body. Since the overall appearance of a human body is influenced by its internal muscle structures, the muscle model is the most crucial layer for realistic human animation.

The muscle layer is implemented as a system of free-form deformation (FFD) [83] lattices in which the skin geometry is embedded. Its only function is to deform the surface's geometry that represents the skin. The deformation of the muscle is controlled through the joint angles and torques generated by the skeleton layer, under the following restrictions: the points of the muscle on the bones must be attached during animation. These deformations will be attached and will deform the geometric skin surface to the skeleton. The FFD are designed to maintain continuity on the outside of the bones and create the skin bend on the inside of the bones.

Finally, the skin or surface layer determines the position of the joints of the skeleton layer and the geometry of the muscle layer. The deformation of the skin layer during an animation occurs by associating the vertices of the skin surface with the joints of the skeleton and the muscle layer.

The key advantage of the layered approach is that once a character is constructed, only the underlying skeleton needs to be scripted to generate the animation. The shape's dynamics is then constructed automatically from the muscle layer. Additionally, the character layer can be used as a template to transfer the motion to other similar characters. Different approaches are based on a more anatomically accurate model of a human character, using bones,

muscles, tendons and fatty tissue in order to simulate a more realistic and articulated deformation of the motion of the human body [84].

5.3 Cyclic animation of human motion

5.3.1 Introduction

A cyclic animation involves the construction of a sequence of frames that can be used for repetitive motions, such as character walking, running or other actions contained in the background of the animation [69]. In this section we propose a modeling technique for producing cyclic motions of human body. The 3D software used in this application for manipulation of surfaces and animation is Maya from Autodesk [9]. Maya is a popular, integrated node-based 3D software suite with an internal scripting language called Maya Embedded Language (MEL) [63]. The surface of the human body has been created from a set of pre-configured curves that were used as the set of boundary conditions to solve a number of PDE. These boundary curves are attached to a skeletal system that holds the animation for cyclic motions. An important function of the method described here is the use of mathematical expressions within Maya software for generating the cyclic motion leading to a very realistic movement. Additionally, with the use of a MEL script User Interface (UI), the user can interactively manipulate the position and movement of various body parts to achieve various cyclic motions or poses. Finally, the animation can be transferred to either the original mesh model from where the boundary curves associated with the PDE surface were extracted or to another mesh model with equivalent topology.

Cyclic motion of a human model is usually achieved by animating first a simple representation of a human body. Usually, this is carried out by using a skeleton system to store and manipulate the animation. This technique, often referred as rigging, uses several “bones” or joints connected to each other to control the overall movement of the skeleton. The motion of the skeleton can be created manually by positioning the handles that control the joints to desired poses and then several interpolation techniques can be applied to generate the in-between positions. More information on these techniques can be found in [85]. A different approach on generating animation is by using expressions that control the movement of the joints which can create very realistic movement [86]. The animation in this work was created with the use of mathematical expressions in Maya that are included as script based instructions which allow the user to control the attributes of the objects. Expressions are useful for linking attributes between different objects where a change in one attribute can modify the behaviour of the other.

Additional steps involve connecting the animation to the human model surface that will be used for the final animation. Once the mesh is skinned with the animation, every transformation applied to the skeleton will be applied directly to the mesh object. Finally a new motion retarget technique is used for retargeting the animation to a different human mesh model, enabling the possibility to have multiple character objects sharing the same animation.

5.3.1.1 Previous work

The problem that arises in animating human models consists of portraying the realism of the movement. A number of previous works in the area have introduced new techniques for achieving realistic motion. These include:

- An automated method for modelling cyclic 3D motion [87] using a new algorithm that enforces smooth transitions between the cycles by operating within the Fourier domain. A key point in this method is its ability to automatically deal with noise or missing data.
- A method for learning and tracking human motion for video [88].This method converts large sets of periodic human motion data automatically into cycles. The learned temporal model provides a prior probability distribution over human motion that can be used for tracking human subjects in complex monocular video sequences and recovering their motion.
- A technique that allows content creators to easily integrate virtual humans into Web3D virtual environments, such as X3D and VRML [89].This technique allows the user to interactively integrate virtual humans acting as formal instructors in virtual environments for learning or training purposes. For example, virtual humans can be used to show and explain maintenance procedures, allowing learners to receive more practical explanations which are easier to understand.
- A three layer hierarchical control system for animating human avatars in 3D virtual environments [90]. The first layer controls the movement of the avatar's joints, the second defines skills or basic behaviours and the third executes a behaviour-based script language that can be used to describe stories to be performed by the avatars.

To date, as far as the author is aware, the majority of techniques for animation of cyclic human motion have only been obtained using mesh models. The use of the PDE method [60] for surface generation provides several advantages for the manipulation of the surface. For instance, once a surface has been initially defined, it may be necessary to manipulate it in order to improve the shape. Hence it is desirable to have control over the shape of the surface once it has been defined. For that reason, the PDE method gives us the advantage to represent a surface analytically, thus making the manipulation of that surface fast and accurate.

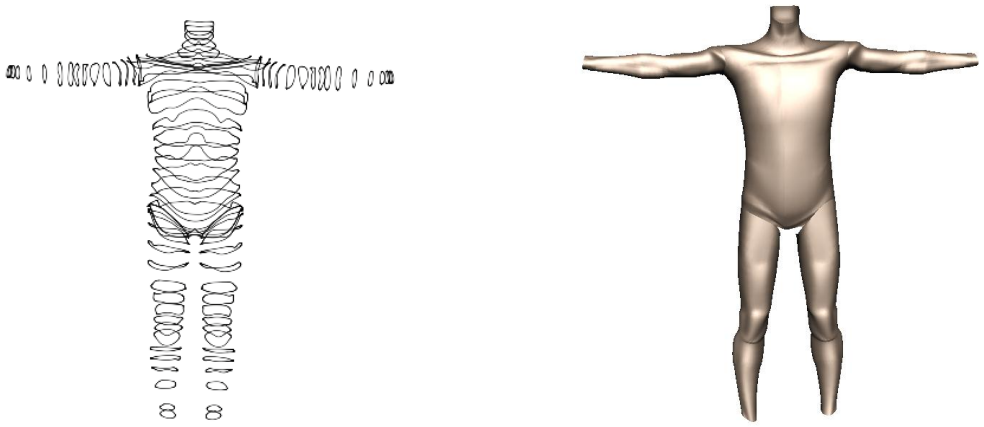
The aim of this technique is to take advantage of the PDE method and use it as the foundation to develop a methodology in human body animation by applying expressions for generating animation with cyclic motions and also retarget the animation to different human mesh geometry. All the examples presented throughout this section have been created interactively in Maya.

5.3.2 PDE-based parameterization of the human skin

The surface representing the outer contour surface of the human body used in this work was obtained through the PDE method. Details on the complete formulation and solution can be found in Chapter 2.2.

Autodesk Maya was used throughout this work to calculate and animate the PDE-based human body. An existing mesh model of a human body is imported to the Maya environment as a guideline to extract the boundary information responsible for generating the surface. The procedure used to extract the curves is explained in details in Chapter 3 Section 4.

Figure 5.3 (b) shows the complete curve-set for all the main parts of the human body that was used to generate the PDE surfaces in Figure 5.3 (a) and (b). The human model is divided here in 5 main regions, torso, left arm, right arm, left leg and right leg. Each region consists of its corresponding boundary curves and the resulting PDE surface accordingly. This surface is a close approximation to the original mesh.



(a) (b)

Figure 5. 3. Complete curve-set of human body. (a)
The resulting PDE surfaces generated from the curve-set. (b)

5.3.2.1 On the construction of an articulated skeleton

Skeleton's "bones" or joints are very important for defining how a character moves [91]. A skeleton system is a hierarchical structure that consists of a series of rigid links connected at joints. The representation of hierarchical model is usually done by a tree structure of nodes linked together by arcs. The highest node of the tree is called the root node. The position of all the other nodes of the hierarchy is located relative to the root node. A node that is higher up in the hierarchy than another node is called a parent node and the one below is called

the child node. The human animation in this work is controlled from a skeleton system organized by a hierarchy of joints. This system defines the human body skeleton to which the bounding curves required for calculating the PDE surface, are attached and animated. The process consists of constructing a skeleton using the human character mesh model as a guideline. In order to generate basic cyclic motions, the articulated figure consists of a series of connected joints representing legs, arms and torso. The skeleton hierarchy controls the movement of the figure and is used to transfer the animation to the “skin” data so that a given pose is produced [92, 93]. The skeleton shown in Figure 5.4 contains the joints and several controls for different parts of the body, each control can manipulate the part of the body to which it is connected to. These controls will be later associated with a series of parameters used to control the motion and position of the joints.

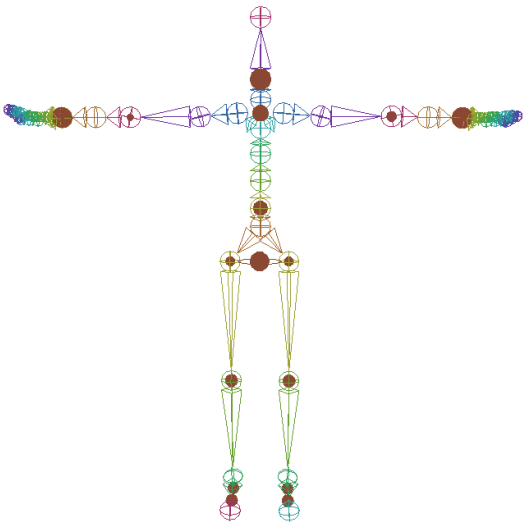


Figure 5. 4. Articulated skeleton system used for PDE-based cyclic animation.

5.3.2.2 Skeleton to skin binding

Once the motion of the skeleton system has been defined, the next step is to attach the skeleton to a “skin” surface. This process is called skinning and is a

mesh deformation technique that can be applied to soft body surfaces as well as other geometrical primitives [94]. The skeleton to skin binding is defined in a geometrical manner. Every vertex of the skin surface is associated with one or more skeleton joints. When a skeleton joint is rotated, the appropriate vertices are deformed to simulate the motion. Skinning a single surface is a fast method; however it can generate several distortions on the surface if not applied correctly.

In this work, the “skin” surface is represented as a set of boundary curves that define the human character model. Given that the skeleton has been defined using the character model as a guideline, the extracted boundary curve data has to maintain the same position in the Cartesian space. Skinning is then performed between the articulate figure and the curve set in the Maya environment by binding each curve-set group to the correspondent skeleton joints. Figure 5.5 shows the boundary curves that represent the human body geometry attached to the skeleton system for animation. Further consideration must be taken when fitting the curves to the joints accurately [95]. Problems can occur if the curves are not aligned properly with the skeleton joints resulting in deformed surfaces as shown in Figure 5.5. The distorted surface in the arm area (shown in Figure 5.5) occurs since the starting point of each curve in the curve-set is not aligned properly, resulting in an unwanted wrapping effect. This problem can be solved during the extraction process of the curves, where techniques capable of sorting of curve points or aligning of curves can be applied. This has been discussed in Chapter 3.

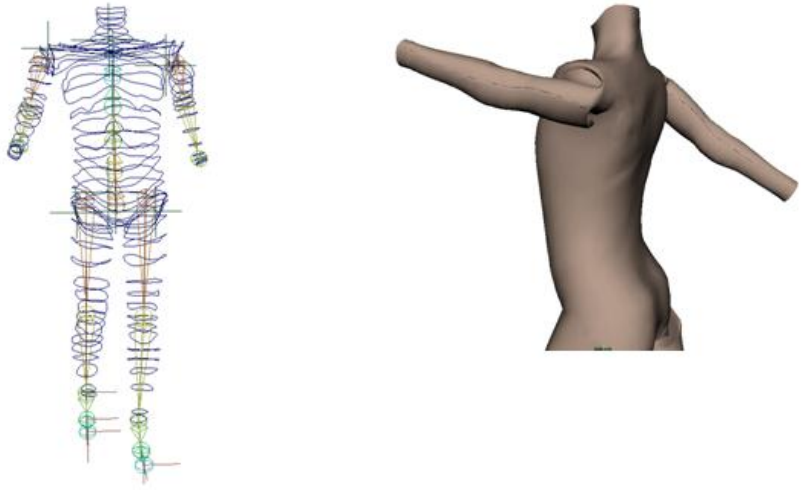


Figure 5. 5. Boundary curves attached to the skeleton (a).
Example of distortions when skinning is not properly applied (b).

5.3.3 Expression driven animation

The next step involves the use of mathematical expressions to re-position the skeleton system for calculating the human based cyclic movement. Understanding the various phases involved in the process of locomotion is crucial to produce realistic human based motion. Various every day actions such as walking, running or reaching for an object can be expressed as cyclic motions. Examining the human walk in more detail, it can be characterized as a sequence of phases separated by foot strikes and takeoffs. The repetition of a phase sequence is called a cycle and the time take to complete is called period. Furthermore, a phase within a cycle can then be defined from a set of parameters responsible for a particular movement. The animation in this work has been produced with the use of mathematical expressions that are included as script based instructions allowing the user to control the attributes of an object within Maya though MEL script. The control is based on a mathematical function, time variables or various attributes of an object. They can be used to

add various constraints to a predefined motion or can directly generate a cyclic motion such as walking or running.

The general motion of the body parts involved in cyclic motion is controlled by periodic functions such as sines and cosines. Let x be a coordinate independent to the described movement, y the coordinate describing the height of the human body and z the direction of the movement itself. Thus, the translation associated with a given cycle motion of any of the body parts involved is given by,

$$\begin{aligned} T_y &= \sin(\omega t) \\ T_z &= \cos(\omega t), \end{aligned} \tag{5.4}$$

where T_y and T_z are functions representing the translation in y and z coordinates respectively, t denotes time and ω the frequency of the movement. Note that as both directions are using the same frequency, their speed is also the same in both directions. The formula for animating the right foot using the above expressions are presented below,

$$\begin{aligned} R_{Fy} &= T_y(H + H_R) + H_{RF} \\ R_{Fz} &= -T_z(L + L_R), \end{aligned} \tag{5.5}$$

where H controls the height of the step and is applied to both feet and H_R and H_{RF} are fine tune parameters controlling the step height of the right foot together with the initial height of the right foot respectively. Note that these two parameters are generally set to zero by default. The L parameter controls the length of the step and is applied to both feet whilst L_R is a fine tune parameter adjusting the length of the step given by the right foot which by default is equal

to zero. The use of a minus sign in T_z ensures that the feet are moving in opposite directions and thus, the code responsible for modelling the movement of the left feet will use Equation 5.5 with a different set of parameter and without the positive sign.

5.3.3.1 Designing movement

The skeleton system consists of several constraints placed in key positions in order to individually control various joints. These constraints are associated with a series of parameters that are used to control the motion and the position of the skeleton structure. An important feature of the PDE-based human cyclic animation technique discussed in this section is represented by a script UI tool that add several functionalities for designing motion [86]. Once the skeleton structure is configured and attached to the boundary curves representing the human mesh model, the user can adjust various parameters linked directly to the skeleton. Figure 5.6 shows the script based user interface (UI) used to re-position the skeleton joints and control the cyclic animation. The user can select the initial speed of the movement and the starting preset of the cyclic animation from the panel. The panel offers three default preset motions that can be adjusted from the various parameters. Figure 5.6 (b) consists of the parameters controlling the Walk and Run cycle. These parameters are linked to skeleton constraints required for manipulating basic characteristics of the motion such as the height and the length of each step or hip and shoulder rotation. Basic as well as advanced controls are available to adjust the cyclic motion or design a new pose for the skeleton system. Once the motion is defined, the animation can be produced by pressing the animation playback button in the Maya

environment. The movement is initialized from these parameters and calculated for each frame of the animation cycle. The UI panel offers great advantages since it is simplifying the process of defining and controlling a human based cyclic motion.

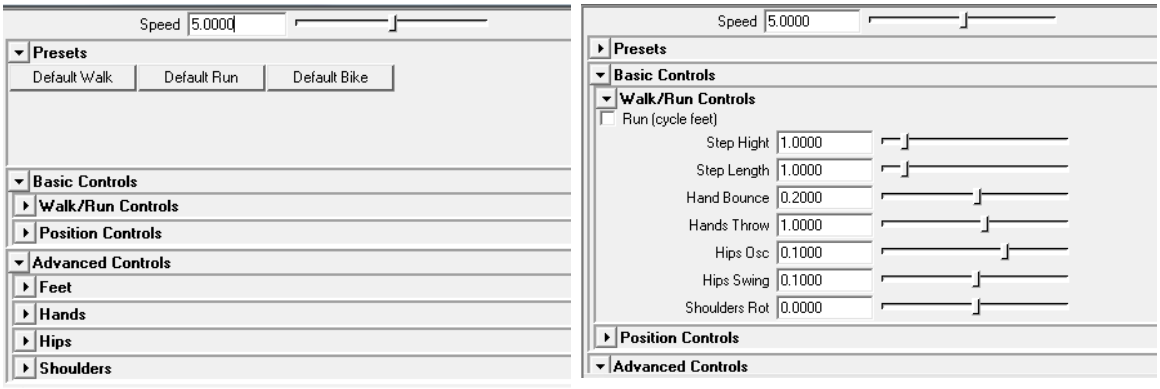


Figure 5. 6.The Maya script based UI used to control the cyclic motion. (a) Controls used to manipulate walk and run parameters linked with the skeleton system. (b)

Finally when an animation cycle has been defined over the PDE representation, the motion can be retargeted to the original human model where the curves where extracted or to a different human based mesh model with similar topology [96]. Given that motion is difficult to generate from scratch using traditional methods for different characters, the use of existing motions is a faster alternative to obtain the animation. Moreover, for each motion retarget process of the animation cycle, the final animated geometry is exported and saved in Maya for playback or further manipulation.

The aim of motion retargeting is to transfer the motion created for one figure to another one with identical structure characteristics (arm length, torso size etc). When two figures share the same structure, the motion of one may not apply to the other and sometimes requires adjustments. Adjusting various parts of the

PDE-based representation model to ensure topological alignment and preserve the motion is an important step. Figure 5.9 shows a case where the armadillo model [6] contains a different topology from the PDE template representation. The torso area for example has completely different topology from the template character representation. Moreover additional features such as tail, nails or body armour cannot be successfully aligned with the default PDE representation without some topological adjustments. The animation of this figure consists of retargeting the motion to the legs and torso area, the top area of the mesh model could not be retargeted without introducing distortions on the surface. The motion retarget technique will be discussed in more detail in Section 5.5.

5.3.4 Results

Figure 5.7 and 5.8 shows a sequence of frames taken from a walk and run cycle using the PDE-based technique discussed in this section for generating cyclic animation. The animation was retargeted to a different model by associating each point of the mesh model with the nearest point of the PDE surface. A motion retargeting technique was then applied to transfer the motion between from one model to another. Figure 5.9, contains a sequence of frames of a walk cycle of the armadillo mesh model. Motion has been applied in the torso and legs region of the mesh only, which is due to the complexity of the armadillo character object. This technique works better with human based characters since the PDE-based representation has been defined from a human character mesh model. A more generic PDE-based template for character representation can be used to eliminate such problems.

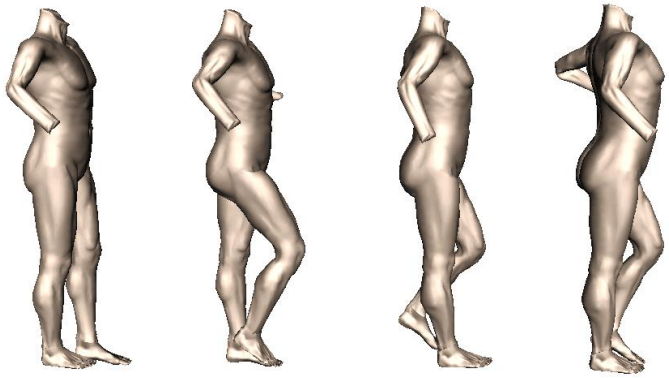


Figure 5. 7. Walk cycle. The motion has been transfer to the original human mesh geometry.

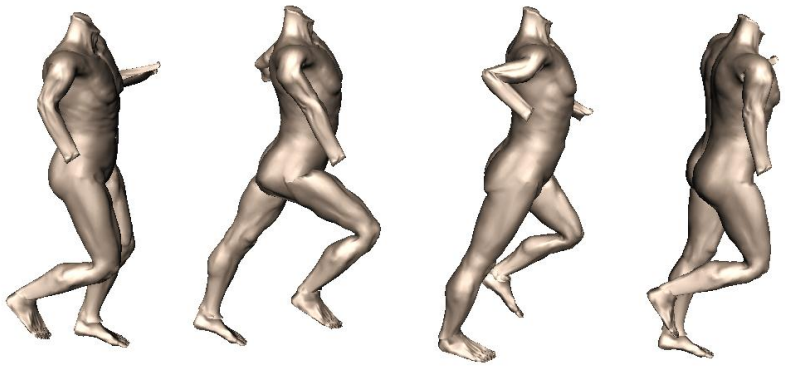


Figure 5. 8. Run cycle. The motion has been transfer to the original human mesh geometry.

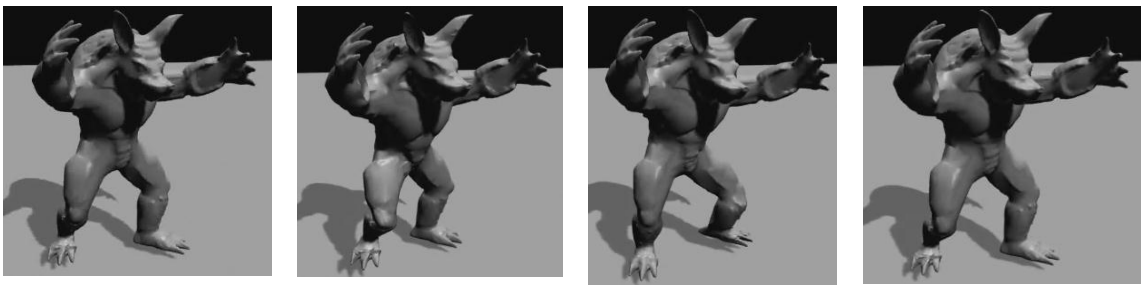


Figure 5. 9. Sequence of frames of a walk cycle of the armadillo mesh model.

5.4 Fish locomotion

5.4.1 Introduction

Another example of animating PDE surfaces is presented in this section. This work is utilizing a fast analytic solution technique for simulating fish locomotion. The spine of the geometric model is fully determined by the analytic solution of the PDE used throughout this work and is exploited for characterizing the wavelike movements observed in fish locomotion. The animation of fish locomotion is a difficult task. It is a subject that has captured the attention of various areas including biology, fluid mechanics [78] and recently robotics and computer animation [76]. In computer graphics and animation a number of factors affect the accuracy of the simulation. Using traditional animation techniques require the manual specification of the motion (e.g. key framing, rigging) that can usually lead to poor results. Moreover, the difficulty to animate the characteristics of the fish locomotion encourages the development of new approaches. Techniques utilizing dynamics animation can add various constraints to a predefined motion as well as automatically generate the fish locomotion. In these models various dynamics theories are used to calculate surface deformation or collision behaviours with another object. Fundamental laws of dynamics can be applied to an articulated hierarchical structure to improve the realism of the motion. Unlike existing techniques for fish locomotion, which are mainly based on spring mass models, the proposed technique is fast and avoids unnecessary numerical errors.

It takes advantage of the mathematical properties related to elliptic PDE surfaces. In particular, it uses the spine of the PDE surface to generate fish locomotion. The spine can be thought as the skeleton or backbone of the geometry of a surface that is controlled through parameters defined on a simple

analytic function. Additionally the fish locomotion technique presented in this work can be applied directly to fish geometry generated using the PDE method or transferred to existing mesh models. In the latter case, the animation is retargeted to a different target mesh model using a point-to-point association. Thus, the user can choose between two different representations of the same animation depending on the end-user application. The aim of this work is to develop a simple technique for simulating fish locomotion. The analytic solutions of elliptic PDEs is used as a tool for achieving such motion since its properties can easily be adapted into an algorithm capable of producing such undulatory movement.

5.4.2 Background

The movement observed in fish when swimming is normally generated by contracting the muscles to produce waves that travel the length of the body [97]. This movement can be divided into two categories the periodic propulsion and transient movements. The periodic propulsion is responsible for the displacement, whereas the second one consists of sudden movements generally associated with sudden changes on speed and direction. These actions usually occur from sensorial identification of a close-by obstacle or predator-prey detection. The full characterization of the undulatory movement observed in most aquatic animals when swimming can be found in [98].

This work divides the periodic motion observed in aquatic animals into four subcategories and identifies which type of fish belongs to each of these categories.

The four subcategories of periodic propulsion are:

- Anguilliform. This movement is considered the most basic form of motion. It is observed in long fish such as eels and tile-fish and distinguishes from the other categories since the whole body length is used throughout the motion.
- Subcarangiform. The undulatory effect is observed in the back region of the fish with the amplitude increasing towards the tail. Trout and cod are examples of possessing this type of motion and it is considered as an intermediate mode between the anguilliform and carangiform mode [ref].
- Carangiform. Examples of fish possessing this kind of movement are sea-bass, barracuda and gold fish. The main characteristic in this movement is the fact that the motion is visible on the posterior half of the body, making this part of the body more flexible than the rest.
- Thunniform. Aquatic animals possessing this movement are considered as the fastest and long distance swimmers. Main characteristic is the crescent shape tail fin, which is responsible for increasing the speed. Fish that fall in that category are shark and tuna.

5.4.2.1 Related work

The work of Tu and Terzopoulos [76] is regarded as one of the first attempts to generate a realistic animation of artificial fish where physical characteristics such as physics, movement, perception and behaviour were included. The system simulates the appearance, movement and behaviour of individual fish as well as complex group behaviours. The animation of the fish locomotion is calculated from a dynamic mass-spring system that can maintain the structural characteristics of the body while allowing flexing. To synthesize realistic fish

locomotion the mass-spring system consists of 23 nodal point masses and 91 springs. The motion is then controlled by four springs called motor controllers and account for the amplitude and speed of the fish movement. The behaviour of the fish is controlled by an autonomous agent which consists of sensors and a brain with motor, perception and learning behaviour. Moreover, most of the work carried out in fish locomotion and virtual environments is based on the spring-mass model as described above. Generally, spring-mass models are solved using numerical techniques that have the tendency to be either slow or prone to numerical errors.

Another example of fish locomotion is presented in [99]; the authors developed a virtual aquarium environment where a mechanics-based fluid expression generator controls the flow of the fluid. The movement of the fish and seaweeds is based on the spring-mass model. Additional factors such as social and collision with external objects have been included in this virtual reality systems. A work for animating fish locomotion in virtual environments using fuzzy logic is presented in [100]. This work describes a framework for animating and controlling articulated bodies in a fluid simulation. This technique relies on the calculation of the fluid-solid interaction forces and a motion control strategy based on fuzzy logic.

5.4.3 Modelling fish locomotion

The process of fish locomotion can be divided in three steps. The initial step consists of constructing the PDE surface representation of the fish as well as identifying the spine and the radial component of that surface. The second step is related to the analytic formulation used to animate the spine of the surface

throughout the animation cycle. Finally, the third step consists of retargeting the motion from the PDE surface to a target mesh model.

5.4.3.1 Construction of PDE geometry

Constructing the PDE representation of a fish can be achieved by obtaining a set of boundary curves that represent a given fish geometry. These set of curves can be manually designed or extracted from within a CAD environment and then used as the boundary data for the construction of the PDE surface. The cross sectional curves are essentially composed of vertices obtained from the original mesh model so that the PDE based surface preserves the realism of the mesh model as accurately as possible. Mesh models of the fish used in this work have been found in [67]. Given that the movement of fish is mainly divided into four major types, at least one fish representing each of these categories has been chosen to be modelled. The fish representing the anguilloform movement is an eel. Fish swimming in subcarangiform are represented by a tropical fish and the dolphin has been selected as a suitable example of thunniform form. Figure 5.10, shows the original geometric model of a dolphin from which a set of 37 boundary curves has been extracted leading to its corresponding PDE surface. Additionally, the spine associated with this PDE surface representation is also outlined. The original geometric model is composed of 2932 vertices and the PDE configuration has been calculated over a grid of 41 by 41 points.

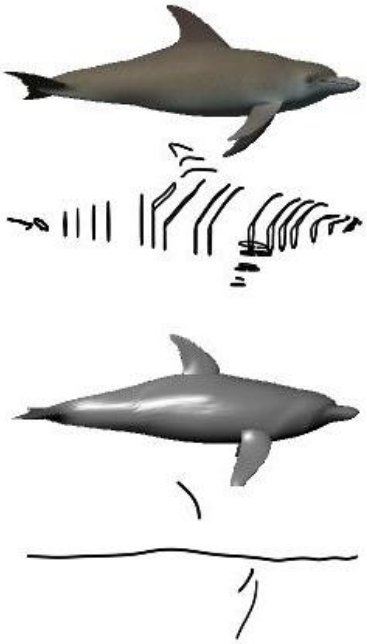


Figure 5. 10. Original geometric model representing a dolphin, the set of extracted curves from the original model together with the resulting PDE surface obtained and its corresponding spine.

5.4.3.2 Locomotion using the spine of a surface

The next stage consists of finding a suitable mathematical equation for modelling the undulatory movement of fish locomotion using the spine of the surface [29]. As mentioned before the spine of an object is can be thought as the skeleton of the geometry of a surface that is controlled through parameters defined on a simple analytic function. Additionally, the spine of a surface is represented as a cubic polynomial curve that is used to control the shape of the PDE surface. The mathematical function will be applied directly to the spine so that it will generate the wave like movement effect on the PDE surface [80]. Moreover, it is desired that the analytic expression provides a certain degree of freedom to the user so that adjusting different parameters will produce different movements. The analytic expression used to manipulate the spine, denoted by $Sp_i(u,t)$ throughout an animation cycle is given by,

$$Sp_i(u,t) = Sp_{original} + \Omega(u)\cos(au+\varphi)\sin(2\pi\omega t_i), \quad (5.6)$$

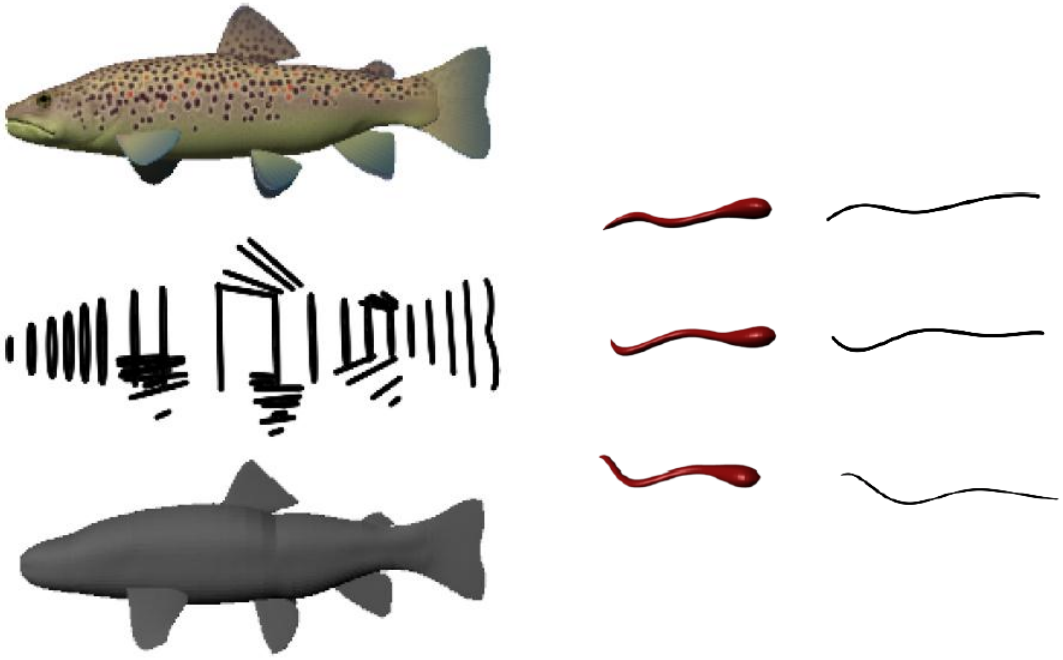
where $\Omega(u)$ determines the amplitude and depends upon u, ϕ represents the phase, α denotes the wave number and ω regulates the frequency of the movement. The subscript i determines the frame for which each animation cycle, and $t_i \in [0,1]$ is the time associated with each respective frame. Moreover, this equation can represent any of the four categories in which the fish movement has been classified. Animation of the PDE surface representation is then achieved by manipulating the spine associated with the model to be animated and adding to each point in the parametric domain its corresponding original radial component for every given frame. Table 5.1 lists the values of all the required parameters to manipulate the spine according to each of the fishes modelled in this work. This table lists the values of all the required parameters to manipulate the spine according to each of the fishes modelled in this work. The parameters are: amplitude $A(u)$, wave number α, phase ϕ and frequency ω.

Movement	Modelled Fish	$A(u)$	α	ϕ	ω
Anguilliform	Eel	$\frac{0.3u}{2\pi}$	1.5	$\frac{\pi}{2}$	1.0
Subcarangiform	Trout	$\frac{u}{2\pi}$	0.75	$\frac{\pi}{2}$	2.0
Carangiform	Tropical Fish	0.1	1.0	$\frac{\pi}{2}$	2.0
Thunniform	Shark	$\frac{u}{2\pi}$	0.25	0	1.0
Thunniform	Dolphin	1.0	0.75	$\frac{\pi}{2}$	1.3

Table 5. 1. Parameters used to manipulate the spine of a PDE surface representing each fish according to its corresponding type of swimming movement.

With this formulation, conservation of volume is not guaranteed. However, if the parameters are set correctly the animated surface will maintain the change of volume within reasonable limits while not affecting the realism of the model. Figure 5.11 (b), shows the original PDE surface representation of an eel fish with two different frames of the animation cycle. The spine associated with each of these PDE surfaces has also been outlined in the Figure. Figure 5.11 presents the original mesh model, the set of extracted boundary curves and its corresponding PDE surface. The fish selected to represent this process has been the trout. Once the PDE surface representation of the models has been found, the spine will be manipulated according to Equation 5.6 using the parameters from Table 5.1. Additionally, the animation of the PDE surface representation of each fish will be obtained over a cycle. The duration of the

cycle has been set equal to one second and twenty-four frames have been obtained during each cycle.



(a) (b)

Figure 5. 11. Original mesh model of a trout, extracted boundary curves and resulting PDE surface representation (a). PDE surface representations of an eel and the spines associated with these surfaces at different times over an animation cycle (b).

Once an animation cycle consisting of n frames has been obtained over the PDE surface representation, the animation can be transferred to either the original mesh model from where the boundary curves associated with the PDE surface were extracted from or to another mesh model representing the same type of fish. The advantage of using this technique is that two different representations of the same object are modelled simultaneously, giving the user the opportunity to choose between a complex and a simplified representation depending on the end application where the animation is required. Figure 5.12 shows the initial and two additional frames of an animation cycle, the motion was computed in the cylindrical PDE surface and then transferred to the fish

geometry. The process of transferring the animation from a PDE surface representation to any mesh model representing the same type of fish will be discussed in more detail in the next section.

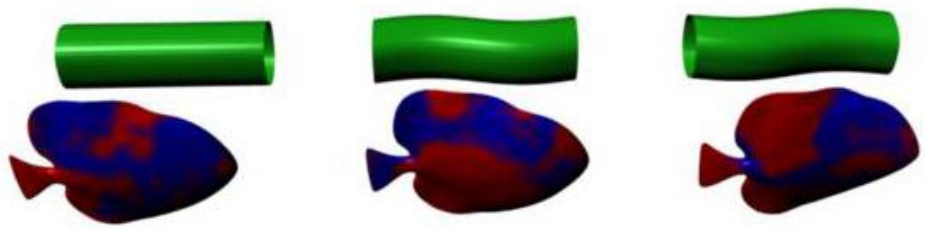


Figure 5. 12. PDE surface representation of cylinder and mesh model of the tropical fish at different times over an animation cycle. The initial configuration and two different frames are displayed here for both the PDE surface and the mesh model.

5.4.4 Results

Figure 5.13 shows a sequence of frames containing different types of fish. Motion has been computed initially in the PDE surface representation of each of these fish using the spine of the surface, and then transferred to the original mesh models. The aquatic environment is a video clip produced in Maya to demonstrate PDE-based fish locomotion. The four different types of fish were modelled and animated by adjusting the parameters obtain by table 5.1 according to the category they are classified.

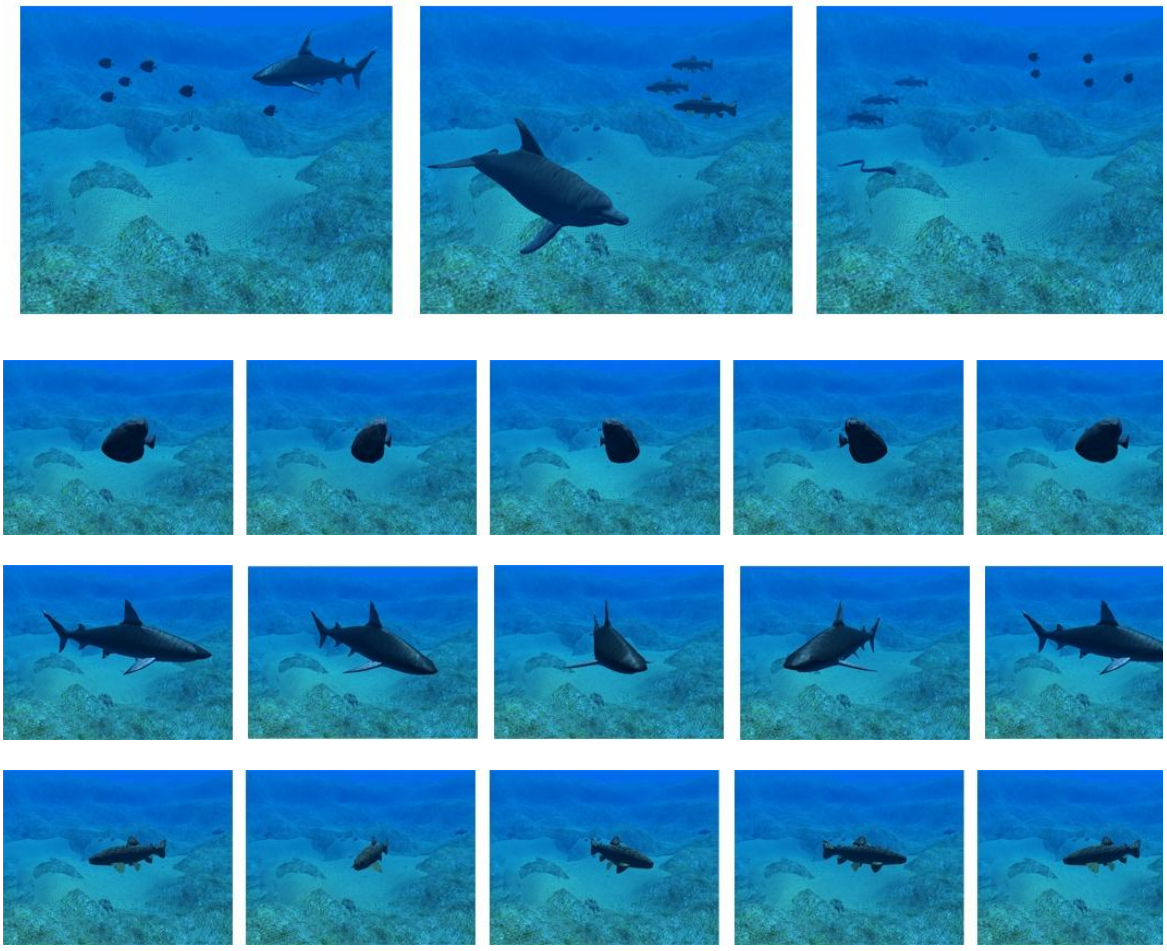


Figure 5. 13. Sequence of frames belonging to an animation cycle of the original mesh models associated with different types of fish. The first, the sixth, twelfth, eighteen and last frame are shown for a tropical fish, a trout, a shark and a dolphin on each case.

5.5 Motion retargeting

In the last two sections a motion retarget technique was introduced and applied directly to PDE surfaces for transferring a cyclic motion to a mesh object. This section will discuss in more detail this technique. Given that motion is difficult to generate from scratch using traditional methods, the use of existing motions is a faster alternative to obtain the animation. A motion retargeting technique consists of simply transferring the existing motion to different target objects in order to achieve the desired effect. In summary, it re-maps animation onto characters with different morphologies.

Generally few techniques address specifically the problem of motion retargeting. Animators usually have to adapt motions for different characters using existing animation tools such as key framing or articulating figure controls. Some commercial systems, such as Maya and Poser, are beginning to support motion retargeting. Some of these new techniques addressing the problem of motion transfer are presented below.

The work developed by Hecker [96] presents a system for animating characters whose morphologies are unknown at the time the animation is created. The proposed tool allows animators to describe motion using existing posing and key-framing methods. At runtime, the generalized data are applied to specific characters to fulfil various pose goals that are supplied to an inverse kinematics solver. This system allows animation of characters with highly varying skeleton morphologies.

Another technique for motion retargeting is presented in the work carried out by Gleicher [101], where the focus is on adapting the motion of one articulated figure to another one with identical structure but different segment lengths. The

method computes the adapted motion for each new character using the space time constraint approach. Given that the technique looks at the entire motion, it can make adjustments in the resulting motion based on all the requirements. Specific features of the motion are indentified as constraints that must be maintained. A constraint solver computes an adapted motion that re-establishes these constraints while preserving the characteristics of the original motion.

In Monzani [102] a new technique is proposed that introduces an Intermediate skeleton to solve the motion retargeting problem using articulated figures. Given a captured motion associated to its Performer Skeleton, the problem of retargeting the motion to the end user skeleton is divided into two steps. The first step is introducing an intermediate skeleton to convert the motion from one hierarchy to a completely different one. The last step corrects the resulting motion and enforces various Cartesian constraints by using inverse kinematics.

The technique of motion retargeting presented in this section is based on a point-to-point association between two objects with similar topologies as seen in Figure 5.14. Motion is then transferred from this association. Once an animation cycle or a required pose has been obtained over the PDE surface representation discussed in previous chapter, the animation can be transferred to the original mesh model from where the boundary curves associated with the PDE surface were extracted or to a different model with similar topology. Note that this technique can be used directly to polygon meshes as well as PDE surfaces; the procedure would be the same since the required data for both surfaces are the vertices that define the mesh models.

The process of transferring the animation from a PDE surface representation to a given mesh model is carried out as follows:

- Alignment of the initial PDE surface and the target mesh model in the same initial position. Key features of the model have to be positioned so that they nearly overlap. This facilitates the correct correspondence between the two surfaces.
- Mapping correspondence between models. This process consists of associating each point of the mesh model with the nearest point of the PDE surface in `their initial configuration'. Thus, each point in the mesh model V_n is represented on the PDE surface by a given point P_n in the PDE surface and it is assumed that this point remain as the closest one to the same particular point at any frame throughout the animation cycle.
- Animation of the mesh model is carried out by finding the difference d_n between the resulting PDE surface at a given frame and the original one for each point. Then, this difference is added to each point of the original mesh model according to the mapping correspondence previously found, Equation 5.7. The procedure can be repeated for any given frame of an animation cycle,

$$P'_n = P_n + d_n, \tag{5.7}$$

where P_n is a given point in the PDE surface, d_n is the difference between the PDE surface at a given frame and the original one for each point.

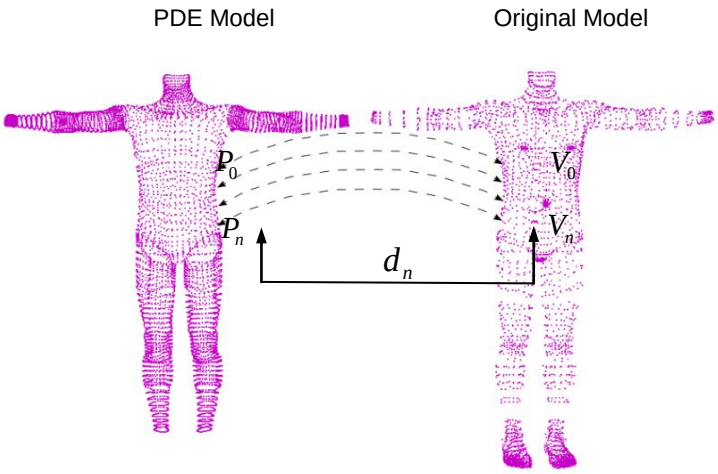


Figure 5. 14.The point to point association between the PDE-based model and the original mesh model.

The advantage of using the PDE-based motion retargeting technique is that two different representations of the same object are modelled simultaneously. This provides the user the opportunity to choose between a complex and a simplified representation depending on storage and memory resources associated with the end application where the animation is required. Note that the quality of the mapping correspondence between the PDE-based representation and the original mesh model depends on the resolution of the grid used to compute the PDE representation. Thus, using a grid with a similar number of points to the number of points in the original mesh will produce a better mapping correspondence. This technique has been used throughout this thesis to transfer motion between a PDE representation and a mesh model. One limitation that could affect the motion transfer is that the quality of the motion is dependant to the point-to-point association of the whole geometry for two given mesh models. This can introduce various distortions on the output surface since accurate mapping between any two given surfaces that don't have similar morphologies cannot be guaranteed. Thus, a segmentation of the surface into meaningful regions will eliminate that problem. In next chapter a variation of this

technique will segment the mesh model in question into vertex maps; each of these maps will be processed separately in order to maintain the motion quality and characteristics.

5.5.1 Examples

Figure 5.15 contains several examples of motion retargeting to various models. The PDE representations, in Figure 5.15 (a, b, c) have been generated using the script based UI panel for re-positioning the skeleton joints. In Figure 5.15, (d, e, f) the animation was retargeted to a different model by associating each point of the mesh model with the nearest point of the PDE surface. Then, the difference between the resulting PDE surface at any given frame and the original one for each point in the surface was found and added to each respective point. Once the point to point association is achieved correctly, motion retarget can be applied to various poses. Figure 5.16, contains examples of retargeting the animation to a different human character model. Note that some features of the character model are not used in the motion transfer. These features are not included in the process since the PDE representation consists of only the basic parts of a human character such as legs, arms and torso. For example, the head of the female character in Figure 5.16 it is not affected from the motion transfer and it can introduce deformations depending on the pose.

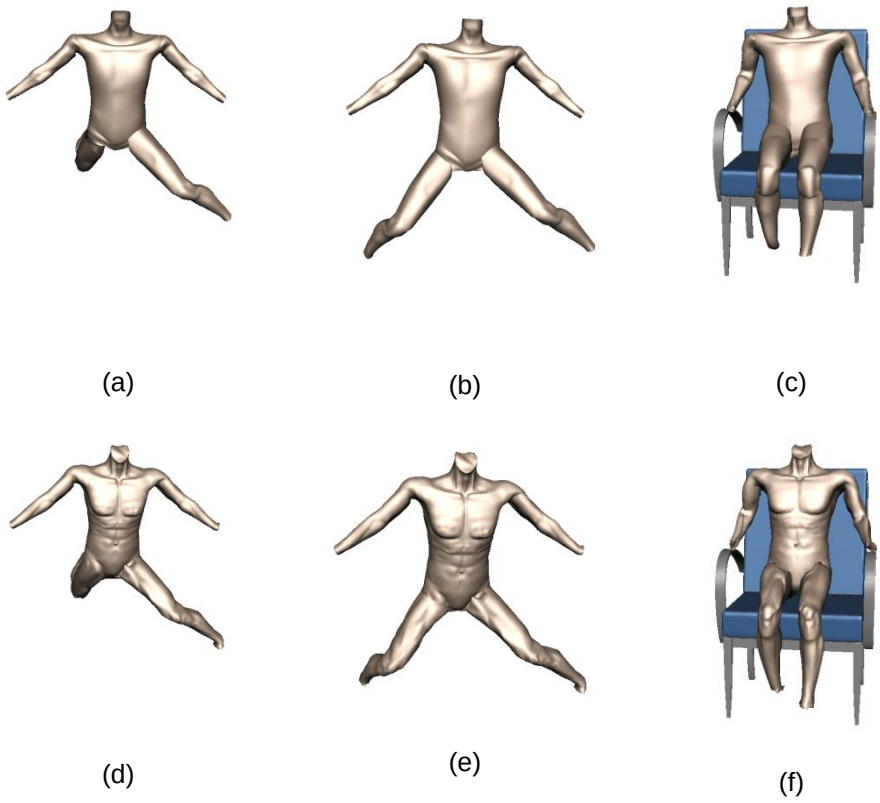
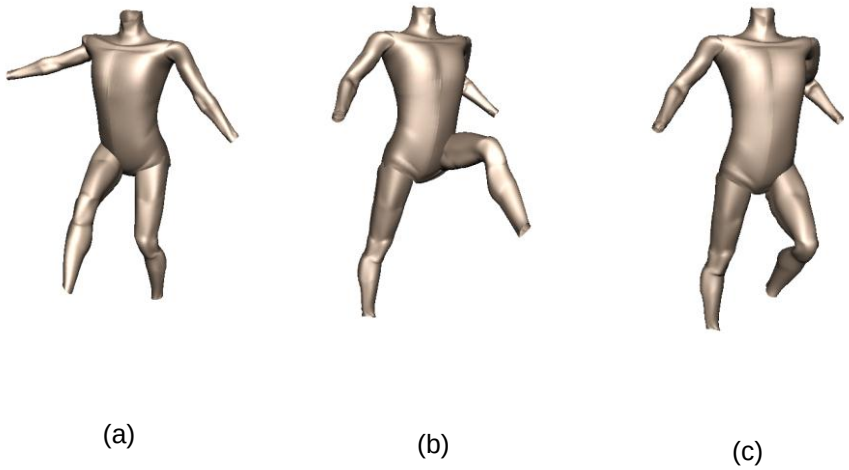


Figure 5. 15. The PDE surface representation of a human model (a, b, c).The resulting animation transfer from the PDE surface to the original model (d, e and f).



(d) (e) (f)

Figure 5. 16. Examples of transferring animation to different model with similar topology. The PDE representation (a, b, c). The transferred animation to the target model (d, e, and f).

5.6 Conclusions

This work presents two techniques for generating cyclic motions using the PDE method. The PDE method can parameterize complex surfaces in terms of a small set of design variables, instead of hundreds of control points. The shape of the surface is then defined through boundary curves and a small set of design parameters. The first technique explores the PDE method in generating every day human-based cyclic motions like walking and running. An important function of this technique is the use of mathematical expressions for generating cyclic motions. By using periodic functions, such as sines and cosines, the general motion of the human body can be controlled, thus making the movement very realistic. Finally, the animation can be transferred to the original mesh model from where the boundary curves associated with the PDE surface were extracted from or to a different geometric model with similar topology. An improvement to be included in this technique is a more generic PDE character representation that will enable the animation to a wider variety of character models. Future work can also be undertaken using spine based

animation to control the movement of the human model, since the PDE method determines the spine of a surface analytically, thus enabling the animation of the entire human body by applying mathematical expressions to the spine as required by any given movement.

The second example presents a methodology to simulate fish locomotion using the spine of a PDE surface representation of a given fish. The analytic solution of elliptic PDE is used as a tool for achieving such motion since its properties can easily be adapted into an algorithm capable of producing such undulatory movement. The spine associated with the backbone of the fish is manipulated analytically to produce a tool for controlling the animation. An analytic function for modelling the undulatory movement observed in fish when swimming was found. A set of parameters associated with the transformation applied to the spine are capable of producing the four categories into which fish movement has been divided. Additionally the motion can be applied directly to the fish geometry generated using the PDE method or transferred to an existing mesh models. In the latter case, the animation is retargeted to a different target mesh model using a point-to-point association. The technique proposed here represents a more reliable method than the already existing spring-mass model since it is faster and less prone to numerical errors. This work can be continued by implementing virtual aquatic environments where perception and behavioural aspects in single or groups of fish can be included. Another future direction consists of applying optimisation techniques for controlling the volume of the surface at each frame of the animation sequence

Chapter 6: Interactive talking head system

6.1 Introduction

Generating realistic facial animation is one of the most difficult tasks in computer animation. The human face is a very well defined structure that allows a large amount of expressions between individuals. Synthesizing facial expressions is an important step in modelling a human face since the face is the main component for communication and personalization of a character [103]. A computer-generated face has distinct advantages, since it is possible to create and control repeatable facial movements. Recent work in facial modelling and animation has been triggered an increase in the appearance of virtual characters in film, video and games. Depending on the end-user application, the model must correspond closely to a specific target character. For realistic facial animation, more complex face models can be used whose surface geometry will correspond closely to a real face.

The geometry and movement of the face is usually associated with the constraints that define the human anatomy. During speech, it is often required to synchronize the movement of the lips and the face geometry. Lip synching involves animating the rigid articulation of the jaw and the muscle deformation of the tongue as well as the lips. Moreover, the animator needs to take into consideration synchronizing the lips with respect to a soundtrack in order to produce realistic speech animation. This Chapter will examine several commonly used facial and speech animation techniques. Additionally, a PDE-based facial animation technique used in a Talking Head system for computer human interaction is also presented.

6.1.1 Facial animation techniques

Significant research efforts have been made in facial modelling and animation since the pioneering work of Parke in [104]. Traditionally, computer facial animation has been divided into three basic steps: Designing the 3D facial mesh, digitizing it, and finally simulate the facial movement. However, over the years a few more steps have been included in the process of producing realistic facial animation systems. An optimal facial animation system or a talking head system will be required to create realistic animation, operate in real time and adapt easily to individual faces. Although some recent work has produced realistic results, the process of generating facial animation still requires human intervention [105]. In this section several techniques used for producing facial animation are presented.

6.1.2 Blend shapes

There are two common techniques used for producing deformable face models: using a blend shape or a physically-based model. Blend shape interpolation [106] is the most intuitive and commonly used technique in facial animation practice; it produces a facial expression as a linear combination of a given set of facial expressions, the blend shapes. A blend shape model can be expressed simply by the linear weighted sum of a number of topologically adjusted shape primitives,

$$u_j = \sum w_k b_{kj}, \tag{6.1}$$

where u_j is the j^{th} vertex of the resulting animated model, w_k is the blending weight, and b_{kj} is the j^{th} vertex of the k^{th} blend shape. The weighted sum can be then applied to the vertices of a mesh model as well as to the control vertices of parametric surfaces [107]. Blend shapes are a standard methodology of commercial animation packages such as Maya, poser and 3ds max. The simplest case of blend shape interpolation is an interpolation between two key frames over a time interval.

Linear interpolation between two key-frames is often used because of its simplicity; however different techniques, such as cosine or spline interpolation functions can be used to produce various acceleration and deceleration effects at the start and end of an animation cycle. In the case of interpolating four key-frames, bilinear interpolation [108] can be used to generate a better variety of facial expressions compared to linear interpolation. Additionally, bilinear interpolation when combined with multiple face expressions, it can generate a wide range of facial expression blending.

One of the main disadvantages in generating facial animation using blend shape interpolation is the blend shape interference. This problem often produces overlapping effects between individual blend shapes. It usually appears in cases where the animator is required to interpolate multiple blend shape poses by adjusting controls associated with these sets of expressions. Multiple blend shape adjustment can re-set a previous desired position of a particular pose. The animator then has to go back and readjust the first desired expression. As a solution, the interference problem can be minimized by processing individual blend shapes locally and by refining these shapes when interference is found.

Computer-based facial animation initially involved shape interpolation, where two or more complete facial expressions are captured and the in-between frames were produced by interpolation. Animation was then accomplished using a cosine interpolation scheme to fill in the intermediate frames between the expressions. The 3D data used to describe the facial expressions were obtained photogrammetrically using pairs of photographs.

The work of Wayters in [109] addresses the problem of automatically synchronizing computer generated faces with synthetic speech. Based on plain ASCII text input, a synthetic speech segment is generated and synchronized in real-time using an articulating mouth and face. A closer interpolated approximation to acceleration and deceleration uses a cosine function to ease in and out of the motion:

Although blend shape interpolations are fast and can easily generate satisfactory facial animations, they are restricted to a small range of facial configurations. Combinations of independent face poses are difficult to produce and often interfere with each other, which cause animators to have to readjust the weights of blend shapes. However, for speech animation, producing fluent natural speech requires calculation of co-articulated visemes. By contrasts to these methods, a physically-based model can be used to simulate realistic speech animation by integrating various skin layers, muscles, fatty tissues and bones on the facial geometry. The speech is then produced by activating the virtual muscles to drive the animation.

6.1.3 Parameterization models

Parameterization techniques are often used in facial animation to overcome some of the limitations of simple interpolations approaches. This is carried out by manipulating a collection of faces through a set of parameters. These techniques allow explicit control of specific facial configurations. A combination of independent parameter values can then generate any possible facial expression with relatively low cost. However, the design of the parameters set is based on manually selecting the vertices and adjusting that a part of the face, which makes the model dependent on the facial topology. A limitation of such techniques is that the choice of the parameter set depends on the facial mesh topology, thus, a generic parameterization is not possible. Moreover, extensive manual tuning is required to set the parameter values.

6.1.3.1 FACS

A widely used facial parameterization method is based on the Facial Action Units (AUs). It was originally introduced by the Swedish researcher Carl-Herman Hjortsjo in 1969, and later extended to FACS, the Facial Action Coding System [110]. According to FACS, the facial behaviour can be divided into 46 action units (AU), each of which is anatomically related to the movement of the facial muscles and jaw/tongue derived from an analysis of facial anatomy. FACS is often used in the muscle based animation systems. Animation using muscle models overcome the limitation of blends shape interpolations and can provide a wide variety of facial expressions.

6.1.3.2 MPEG-4 facial animation system

An extension to FACS is the MPEG-4 Facial Animation standard [111]. It contains a standardized syntax for the definition and animation of synthetic faces by directly manipulating feature points of the face. MPEG-4 animates a 3D face model by defining, a total of 84 face definition parameters (FDP) and 68 facial animation parameters (FAP). The facial definition parameters are responsible for describing the movements of the face, while the facial animation parameters are designed to reproduce expressions, emotions and speech pronunciation.

There are 66 FAPs and 2 high-level FAP (expressions, visemes) defined in total. Each FAP describes which facial points are affected, the direction and the amount of displacement. The 68 parameters are categorized into 10 different groups related to parts of the face. FAPs represent a complete set of basic facial actions including head motion, tongue, eye, and mouth control. This allows the representation of most natural facial expressions. Since the FAPs are required to animate faces of different sizes and proportions, the FAP values are defined in face animation parameter units (FAPU). This unit is based on face model dimensions and can be computed based on some special key points of the face (like eye distance or mouth size). The FAPU's are computed from spatial distances between facial features on the model in its neutral pose. Additionally, the standard specifies the facial animation table (FAT) that determine which vertices will be affected by a particular FAP. Note that the standard does not specify any particular technique for achieving facial mesh deformation of a given FAP. Implementation details such as the resolution of the mesh, deformation algorithm and rendering are left to the implementation of the facial animation system. Various techniques for facial deformation have

been developed over the years. They are categorized below based on the mechanisms by which the geometry of the face is manipulated.

6.1.4 Free-Form deformation model

Free-form deformation (FFD) deforms volumetric objects by manipulating control points arranged in a 3D cubic lattice [83]. A good physical analogy for FFD is to consider a mesh or a group of mesh objects embedded in an imaginary and flexible control box containing a 3D grid of control points, as shown in Figure 6.1. As the control box is deformed into arbitrary shapes, the embedded object deforms accordingly. Mathematically, the FFD is defined in terms of a tensor product trivariate Bernstein polynomial. Alternately, this deformation can be formulated in terms of other polynomial bases, such as tensor product D-splines or non-tensor product Bernstein polynomials.

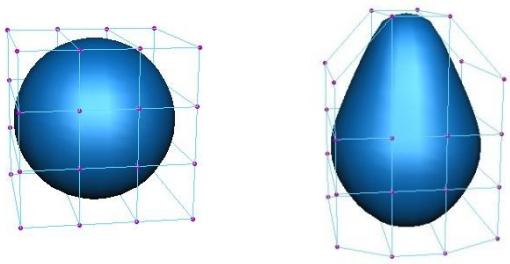


Figure 6. 1. Free-Form Deformation using a 4x4x2 grid lattice

FFD is an efficient and intuitive technique that is independent of the surface geometry. It can deform many types of surface primitives, including polygons, quadric, parametric, and implicit surfaces. One of the main disadvantages is that the shape of the FFD lattice doesn't allow arbitrarily shaped deformation.

However, variations of this technique offer several advantages on the design and manipulation of the lattice. Extended FFTs [112] allows manipulation of the lattice before its association with the mesh object. The lattice can be represented as any geometrical shape (prismatic, cylindrical, tetrahedral, etc).

Another technique is presented in [113] using parametric and implicit PDE surfaces to define geometric solid models. The models will contain both geometric information and intensity distribution associated to the flexible boundary conditions. This technique allows designers to manipulate PDE surfaces of complex geometry through direct sculpting and FFD modelling. Manipulating a FFD control point can be considered as evaluating a physical-based muscle. Additionally, adjusting the position of the control points of the lattice, Figure 6.1, provides a more intuitive and efficient approach for manipulating muscles. However, compared with muscle deformations, the physical-based muscle model that will be covered in the next section provides precise simulation of the actual facial muscles and skin deformation.

6.1.5 Physical-based muscle deformations

Physics-based muscle models are used to create models based on simplified structures of bones, muscles, skin, and tissues. Such models offer the advantage of manipulating facial geometry based on simulating the characteristics of the facial muscles. Physics-based muscle models are grouped into three categories: mass-spring systems, vector representations, and layered spring meshes.

The mass-spring system model for simulating facial animation was introduced by Platt and Badler in [114], [115].Their system integrates FACS notation to

indentify facial regions and consists of 38 regional muscle blocks interconnected by a spring network. The effects of the movement are modelled according to the deformation of the muscles and their effect on the skin. Since facial deformation is modelled through muscle movement, a system needs to describe visible facial expressions at the muscle level. An action unit (AU) describes the action produced by one or a group of muscles in the spring network.

The muscle vector model was originally proposed by Waters in [116]. In this method, a muscle definition includes a vector direction, an origin, and an insertion point. The facial deformation is influenced by the muscles beneath the skin. This model can produce natural human emotions such as anger, fear, surprise, disgust etc by utilizing dynamic parameters that emulate the primary characteristics of facial expressions.

Positioning of vector muscles into anatomically correct facial areas can be a difficult task since incorrect placement can potentially result in undesirable deformation of the facial areas. However, the vector muscle model is widely used because of its representation and independence of the facial mesh structure. An example of this technique is presented in [117]. The system automatically generates 28 facial feature parameters from an input video recorded face. These parameters are then decomposed using the linear model into the shape and action parameters. The animation of the face is approximated by utilizing the Least Squares Fitting (LSF) procedure for the shape and action parameters. The action parameter is then used to drive the animation.

Terzopoulos and Waters introduce the layered spring model in [118], in this work, they proposed a facial model that represents the anatomical structure and dynamics of the human face. The systems consist of three layers of deformable mesh that correspond to skin, fatty tissue, and muscle tied to bone. Each mesh node is connected with the corresponding layer through an elastic spring. The deformation is produced from muscle forces applied to the mesh system. Realistic effects such as wrinkling around the mouth are computed from constraints that are preserving the volume in the tissue layer.

Another example of the layered muscle technique is presented in [119]. This muscle model uses three conceptual layers: a skin / tissue layer, a layer of muscles attached to the skull and the underlying bone structure, composed of the skull and rotating jaw. Implicit surfaces are used for specifying the shape of the muscles. The skull geometry and the layout of the facial muscles are created semi automatically based on the facial geometry. The influence of muscle contraction onto the skin is simulated using a mass-spring system that connects the skull, muscle, and skin layers of the facial model.

6.2 Taking head systems

6.2.1 Introduction

Speech is treated in a different way compared to the animation of facial expressions, key-frame-based techniques provide a poor approximation in representing speech animation [120]. A phoneme is the basic unit of the acoustic speech. A visual representation of the phoneme is called viseme. In speech animation visemes are often used to represent the position of the lips, jaws and tongue in a particular phoneme. This technique is very popular for generating realistic speech without having to manually set the key frame positions for every viseme expression. In virtual environments, it is necessary to keep the data transmission size as small as possible for real time performance. Facial animation usually requires a large set of expressions to produce any given text. Facial data need to be as small as possible to generate the speech animation without any lag or de-phase over the network

A talking head system is an environment where a 3D human head is talking and interacting with a user [121, 122]. Such a system, needs to provide all the necessary tools for creating a computer-human interaction process. To that extend, a text-to-speech (TTS) system is required to generate a sequence of phonemes from an input text. Many phoneme sounds are visually ambiguous when pronounced. Therefore, one viseme can be used to represent several phonemes. The conventional lip sync technique consists initially of decomposing the speech into a set of phonemes. These phonemes will be visually represented in the system as a set of visemes. A mapping between the phonemes in the speech signal and the visemes in the database is carried out to construct the appropriate lip shape. Additionally, the system requires a real

time response to a user's input. Real time dialog systems can include personality, emotions and interactive dialog in a computer-human or computer-computer situation. The process of human-computer interaction is facilitated with the use of chatterbots where actions such as personality, emotions and respond can be integrated within the talking head system. Moreover, emotional tags embedded in the dialogue database can be used to generate facial expressions. The PDE-based talking head system is an application developed in C# using the OpenTK API for rendering and visualization. The application is also utilizing the Microsoft TTS engine for text to speech conversion and the Rebecca AIML chatterbox API for processing user's input.

6.2.1 Related work

The problem arisen from facial animation consists of portraying the realism of the movement. The natural contact with facial expressions and the availability of better and more powerful hardware demand an ongoing improvement of the animation techniques for facial animation. A number of works in the area previously have introduced new techniques for achieving realistic motion. These include:

A Talking Head System for Korean Text [123]; the system animates the face of a speaking 3D avatar in such a way that it realistically pronounces the given Korean text. The proposed system consists of SAPI compliant text-to-speech engine and MPEG-4 compliant face animation generator. The input to the engine is a Unicode text that is to be spoken with synchronized lip shape. The TTS engine generates a phoneme sequence with their duration and audio data.

The TTS applies the co-articulation rules to the phoneme sequence and sends a mouth animation sequence to the face modeler.

Greta: A Simple Facial Animation Engine [124] is a 3D facial animation engine compliant with MPEG-4 specifications; the aim in this work was to simulate in a rapid and believable manner the dynamics aspect of the human face. Greta, a 3D proprietary facial model with the look of a young woman, is the core of an MPEG-4 decoder and is compliant with the “Simple Facial Animation Object Profile" standard. The 3D model uses a pseudo-muscular approach to emulate the behaviour of face tissues and also includes particular features such as wrinkles and furrow to enhance its realism. Facial features such as wrinkles have been implemented using bump mapping which allows a high quality 3D facial model with a relative small polygonal complexity.

Real-time Lip Synchronization Based on Hidden Markov Models [125]; A lip synchronization method that enables re-using of training videos when input voice is similar to training voice sequences. The face sequences are clustered from video segments, then by making use of sub-sequence Hidden Markov Models, the system builds a correlation between speech signals and face shape sequences. This decreases the discontinuity between two consecutive output faces and obtains accurate and realistic synthesized animations. The system can synthesize faces from input audio in real-time without noticeable delay. Since acoustic feature data calculated from audio is directly used to drive the system without considering its phonemic representation, the method can adapt to any kind of voice, language or sound.

Movement Realism in Computer Facial Animation in [126]; This is another work targeting realism in movement and behaviour of agents or avatars in virtual

environments. The system is using co-articulation rules for visual speech and facial tissue deformation producing expressions and wrinkles.

6.2.2. On the development of a PDE-based talking head system

In this section a new technique is presented that re-uses facial animation data for different 3D human face target models. The system uses a set of pre-configured viseme poses to generate a series of PDE-based template models. Each new viseme pose is then retargeted to a different facial mesh model for producing the speech animation. This is a pre-processed operation that takes place at the loading time of the application and it is repeated for all the required visemes in the database.

The 3D human head is synchronized with a TTS engine to generate voice; a text-to-visemes function generates and returns in real time the current visemes of a given input text. The system also integrates an AI bot engine to determine the reply the talking head will give to a given text input by the user. The engine used in this work is the Rebecca (Artificial Intelligence Markup Language) AIML library that implements the Alice bot language processing chatterbox [127]. Chatterbots are computer systems that can produce a human-computer interaction in natural language. The user can enter a question or phrase and the AI bot will generate the appropriate answer to facilitate a real time conversation. Text response from the bot is then captured by the TTS engine and converted to a set of visemes to synchronise and animate the 3D human face. Animation is carried out by linearly interpolating a given set of visemes to generate the in-between transition of the speech animation.

6.3 Text-to-speech

Speech is often more efficient than written messages. Various important developments in speech synthesis and natural language processing techniques resulted in the concept of text-to-speech synthesis. A text-to-speech synthesis can be defined as automatic production of speech through a grapheme-to-phoneme transcription of the sentences to produce [128]. The applications of a TTS system are numerous and extend in various fields. Some of them are:

- Human computer interaction, e.g. Talking head system combines the TTS with a facial animation engine to naturally reproduce a character's facial expressions while reading a specific text.
- Voice narrator, e.g. speech synthesizers in measurement or control systems.
- Disability aids, e.g. speech synthesis for blind users.
- Telecommunications services, e.g. voice support for applications.

Figure 6.2 contains the general functional diagram of a synthesizer. It consists of a Natural Language Processing module (NLP), capable of producing a phonetic conversion of the text read together with the desired voice tone and rhythm (also called prosody), and a Digital Signal Processing module (DSP), which transforms the symbolic information it receives into speech. Phonetic and prosody information together can generate the symbolic linguistic representation that is output to the front-end application. The back-end application, usually provided from the API, converts the symbolic linguistic representation into sound usually referred as the synthesizer. Details about the

architecture of a TTS system is outside the scope of this thesis, additional information can be found in [128].

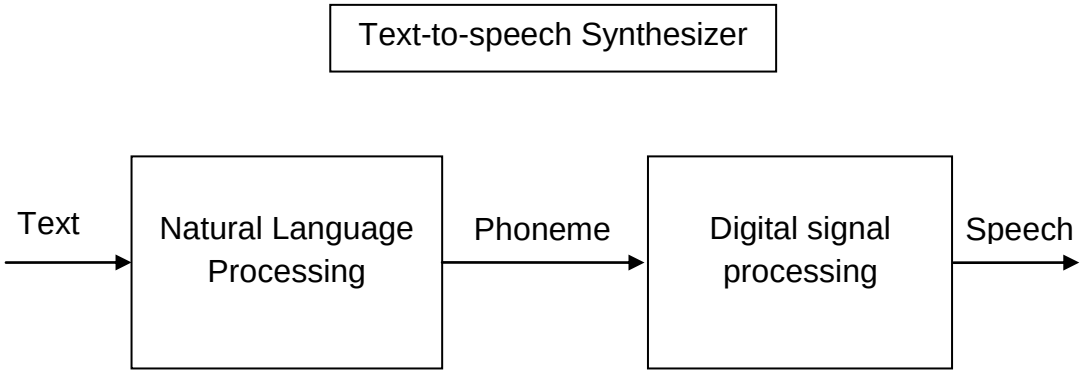


Figure 6. 2. General function diagram of TTS system.

The text-to-speech system converts normal language text into speech; it recognizes the input text and, using a synthesized voice, chosen from several pre-generated voices, speaks the written text. The TTS engine used for this work is the Microsoft TTS which is fully programmable from the Speech API 5.0. The SAPI acts as a software layer that allows speech-enabled applications to communicate with both speech recognition and TTS engines. Additionally, the SAPI is responsible for a number of functions in a speech system, such as:

- Controlling audio input, whether from a microphone, files, or custom audio source.
- Converting audio data.
- Storing audio and serializing results for later analysis.
- Using SAPI grammar interfaces and loading dictation.
- Performing speech recognition.

The speech API provides a high-level interface between an application and the speech engine. All the low-level details needed to control the real-time operations of various speech engines are implemented in a collection of libraries and classes. With the help of the API, the TTS operation, which is required to convert a given text into a set of viseme, can be performed in real time. The text-to-viseme process is used to identify which letter from the database is required to animate a given text and synchronize it with the voice. Using events, the application synchronizes the output speech with real time actions such as the phonemes or visemes generated from the input text.

In this case, the generated viseme is used to query from the database the current viseme index that is required for the speech animation. Table 6.1 contains the first 10 indexes used to identify which visemes are needed to simulate the speech animation. The TTS engine contains 22 visemes in total used for reproducing the sounds from any given input text in English. Note that many phoneme sounds are visually ambiguous when pronounced. Therefore, one viseme can be used to represent several phonemes. For example letters F and V can be reproduced by using the same viseme. Consonant letters such as B, M and P can represented as well with the same viseme. To that extend, the animation system can link various similar visemes together to minimize both loading and processing time.

A total of 15 visemes are used in the talking head database to simulate speech. The remaining 7 visemes from the TTS engine have been associated with other similar visemes. For example, SP_VISEME_1, 2 and 3 from table 6.1 can be represented as SP_VISEME _1.

SP_VISEME_1,	ae,ax,a
SP_VISEME_2,	aa
SP_VISEME_3,	ao
SP_VISEME_4,	ey,eh,uh
SP_VISEME_5,	er
SP_VISEME_6,	y
SP_VISEME_7,	w,uw
SP_VISEME_8,	ow
SP_VISEME_9,	aw
SP_VISEME_10	oy

Table 6. 1. List of the first 10 viseme indexes association.

The integration of the TTS system in the talking head environment was developed using the .NET framework. Using Component Object Model (COM) **ISpVoice** interface applications can control the TTS functionality. Initialization of the engine is achieved simply by creating a **spVoice** object, whereas for text to speech synthesis a single call to **ISpVoice.Speak** is required. Additional functionality for manipulation of voice and synthesis is provided; various function calls can control the speaking rate, the output speech volume and the current speaking voice.

The speak method can be operated synchronously or asynchronously, an important feature for synchronizing the speech output with the rendering API as well as adjusting speech properties in real time. The SAPI communicates with the applications by sending events using standard windows callback mechanisms. Applications can then synchronize to real-time actions such as word boundaries, phoneme or viseme boundaries. Events are used for synchronizing the output speech. Each audio stream generated from the SAPI engine contains an event id where the application can identify position and

status of the stream. A viseme event id is associated with the list in Table 6.1 in order to find the current viseme. This information will help to calculate the interpolation time between subsequent viseme. Additionally, the TTS engine will playback the audio stream from the input text.

Communication between the application and the SAPI is processed as a two step operation. The application first receives a general window message from the SAPI. This message is similar to other window messages used by the operating system, such as mouse, keyboard, and window events. The second step in the communication process consists determining which action occurred. The application needs to determine the exact action that is taking place. A list of all the SAPI defined action can be found using the **SPEVENTENUM** list. Using SPEVENT and GetEvents method the SAPI can identify specific information about the current event. Actions such as **START_INPUT_STREAM, END_INPUT_STREAM** and **VISEME** are used to determine the current activity that is taking place. Once the current event is determined, the application needs to take the necessary action. When a **START_INPUT_STREAM** event is captured, the input stream from a Speak call begins synthesizing to the output, whereas an **END_INPUT_STREAM** is identified as the end of the Speak event. In the case of the VISEME event id, the SAPI returns the relevant viseme with some additional information for the current Speech operation.

6.4 A.L.I.C.E's brain

6.4.1 Introduction

Once the speech engine is configured to capture and process real time events, the following step consists of integrating an Artificial Intelligence Markup Language or AIML chatterbot to generate response from a given text. A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), also referred to as Alicebot, or Alice, is a language processing chatterbox based on an experiment specified by Alan M. Turing [129] in 1950. A chatterbox is a program that engages in a conversation with a user by applying some heuristical pattern matching rules to the human's input [105, 130] A.L.I.C.E. software utilizes AIML, an XML based language used for creating chat robots.

It contains a class of data objects defined in the XML specification [131] called AIML objects and describes the behaviour of computer programs that process them. Various Alicebot clones have been created based upon the original implementation of the program and its AIML knowledge base [132]. Some of these AIML interpreters are:

- Rebecca AIML (C++, Java, .NET/C#, Python)
- Program D (Java)
- Program R (Ruby)
- Program O (PHP)

This work uses the Rebecca AIML library for generating real-time responses. The system establishes a local connection with the Rebecca AIML bot to process and generate response to a given input text. The response is then handled as an input text for the SAPI text-to-speech engine.

6.4.2 Previous work

Persona-AIML [133]. This work presents the Persona-AIML architecture for the creation of chatterbots in AIML (Artificial Intelligence Markup Language) with personality. Computational models of personality are in general adapted from some psychology model or theory. The Personality Component defines the beliefs, the personality elements, and the rules that determine chatterbot's behaviour.

Another work is presented in [134] demonstrating an emotional MPEG-4 compliant talking head system based on AIML. It uses Alicebot to generate response and emotion from given input text. Emotions are embedded in the AIML database as a set of predefined emotion tags. These emotional tags are passed to the personality model to simulate believable behaviors. The personality model, depending upon the current mood and the input emotional tags, updates the mood. Depending upon the output of the personality model, mood processing is done to determine the next emotional state. This process determines the probabilities of the possible emotional states. Additionally, the system generates lip sync from the visemes generated from the Text-To-Speech engine.

TQ-Bot [135]. This work presents an Intelligent Tutoring System using AIML with the aim to provide personalized instruction to students. The authors have developed an open e-Learning platform for helping the students during their learning process and to support the activities of the teacher. The bot is able to analyze the requests made by the learners in written natural language and to provide adequate and domain specific answers orienting the student to the right

course contents. Additionally, TQ-Bot is able to track and supervise the student progress by means of personalized questionnaires.

The brain of A.L.I.C.E. consists of 41,000 elements called categories [127]. Each category contains a question and answer, called the “pattern" and “template". The patterns are stored in a tree structure managed by an object called the Graphmaster, which implements a matching algorithm. Graphmaster matching is a special case of backtracking, depth-first search. In most cases matching is handled by a linear traversal of the graph from the root to a terminal node.

The AIML architecture allows the use of different models of personality in the construction of chatterbots. It implements various tags to introduce randomness in answers, and to keep track of small dialogue history. Although it does not use any syntactic or semantic language analysis techniques to generate the response, the content embedded in AIML is enough to engage the user in believable conversation to a certain degree. A.L.I.C.E contains a learning mode, called supervised learning since a botmaster is required to create and manage the content. The botmaster will monitor the bot conversations and can create new AIML content to make the bot responses more believable, accurate or human like. Moreover, every AIML object has both a logical and a physical structure. The physical structure consists of units called topics and categories, while the logical structure is composed of elements and character references.

6.4.2 AIML elements

The basic unit of knowledge in AIML is called a category [127]. The term category was borrowed from pattern recognition theory [6]. Each category contains of an input question, an output answer, and an optional context. The question is called pattern, the response is called template and lastly, the optional context is divided into two types called “that” and “topic”. The AIML pattern tag consists only of words, spaces, and the wildcard symbols. Words must contain only letters and numbers separated by a single space whereas the wildcard characters can function like words. A pattern element must appear in each category and it must always be the first child element of that category element. A pattern does not have any attributes.

Additionally, AIML supports interaction with other languages and systems. For example the <system> tag can be used to execute any program or command accessible from the operating system and insert the result in the reply. Alternatively, the <javascript> tag can be used to allow scripting inside the templates. The <template> tag is the most basic AIML tag and is always paired with a <pattern> tag. It always appears within <category> elements and it doesn’t contain any attributes. The <template> must follow the <that> element or follow the <pattern> element. The majority of AIML content is within the template. The template may contain zero or more AIML template elements mixed with character data.

Figure 6.3(b) shows an AIML example code, the pattern defines the input and the template tag defines the bots response to that input. The syntax of an AIML category is:

```
<aiml>
    <category>
      <pattern> Input Text </pattern>
      <template> Response </template>
    </category>
</aiml>
```

(a)

```
<aiml>
    <category>
      <pattern> Hello </pattern>
      <template> Hi! How are you? </template>
    </category>
</aiml>
```

(b)

Figure 6. 3. General syntax of an AIML category tag. (a) Simple example of input and output response. (b)

The above AIML code matches the client text input, in this case the word “Hello” and sends back to the client the response “Hi! How are you?”.

The optional context in the category tag consists of two elements, called <that> and <topic>. The <that> element appears inside the category, and its pattern must match the bot’s last response. The <that> element is a special type of pattern element used for context matching. It is optional in a category, but if it exists it must occur no more than once, and must follow the <pattern> and <template> element. Remembering the last response is important for creating a more believable conversation. In the example below, Figure 6.4 (a), the <category> element is activated when the client says yes. The bot must find out what was the question to that answer. If the bot asked, "Do you like AIML?”, the category matches the <that> element and it continues the conversation using the <template> response element.

```
<category>
  <pattern>YES</pattern>
   <that>DO YOU LIKE AIML?</that>
   <template>WHAT IS YOUR
FAVORITE INTERPRETER?
</template>
</category>
```

(a)

```
<topic name="AIML">
 <category>
   <pattern> * </pattern>
    <template>  MY FAVORITE
INTERPRETER IS RebeccaAIML
    </template>
 </category>
```

(b)

Figure 6. 4. <That> element example. (a) <Topic> element example. (b)

The <topic> is an optional element that might appear outside a <category> element, and it is used to group together categories. The <topic> element allows the bot to store duplicate patterns in different topics, this way the bot can generate different responses to the same input patterns depending on the topic. A <topic> element has a required name attribute that must contain a simple pattern expression. A< topic> element may contain one or more category elements. The botmaster uses the <set_topic> tags to set the topic of current <category> element. Once the topic is set, for any new query from the client the bot will start looking for a response in the categories that match the current<topic> tag. If there is not a category defined in the current topic, then any categories that are not defined in topic tags are searched.

Figure 6.4 (b), contains an example of using the <that> element. In this case, if the client says something that the bot does not have a specific response for, it could still respond within the current topic. For example, the response for any undefined pattern element under the AIML topic will be “My favourite interpreter is RebeccaAIML”.

AIML consists of various elements offering leaning capabilities and intelligent response to achieve realistic human computer interaction. Moreover, the <random> element tag can generate different responses to the same input text. Each possible template response for the current pattern element needs to be separated with the <li> tag element. AIML is extensible; the botmaster can include an infinite number of new tags for application specific properties. Predicate tags can be used according to a client based “set” and “get” method to generate an endless variety of responses. Recursive categories can be used to map one input to another one, either for language simplification or to identify similar patterns. The AIML implementation for recursion is the tag <srai>. Figure 6.5 (a), shows a basic recursion example, if the user says “HI”, “Hello”, “Hi there” etc, the response template will be the same as for the “HI” pattern element.

```
<category>
 <pattern>HI THERE! </pattern>
<template><srai>HI</srai>
 </template>
</categroy>
```

(a)

```
<category>
 <pattern>WHAT IS * </pattern>
<template><srai>WHAT IS<star/>
</srai>
 </template>
</categroy>
```

(b)

Figure 6. 5. A basic recursion example (a).
Simplification of input pattern using recursion and the <star> element (b).

Recursions are useful for a variety of tasks, some of these include:

- Symbolic Reduction: Reduction and simplification of complex input patterns.
- Divide and Conquer: Split an input into two or more parts, and combine the responses to each.
- Synonyms: Generate different ways of saying the same thing to the same reply.
- Spelling or grammar corrections.
- Detecting keywords anywhere in the input.
- Conditionals: Branching can be implemented with the <srai> operator.

A common application for recursive categories is simplification and reduction of complex input patterns. A combination of the <star> tag and <srai> recursive calls can be used to produce endless combinations. For example, Figure 6.5 (b) shows an example of recursion using the <star> tag. The bot will match a pattern starting with “What is” and use the “*” value to define a recursion call to find the best match for the transformed input.

6.5 Building the talking head system

In a Data-driven facial animation system usually the expressions are pre-configured and blended together to produce a sequence of letters; whereas the 3D human head is synchronized with a text-to-speech engine to generate speech according to a set of phonemes for a given word. The system is usually interactive and can incorporate changes of emotions dependant on the user’s input. The aim of this paper is to build a viseme-driven talking head system

where a given human head mesh model with similar topology can be animated without the need to generate all the necessary expressions for the operation. The overall process consists of generating a set of PDE-based expressions that are used internally as the template expressions for any given human head model. The PDE method has been used for generating the surface template expressions; thus utilizing the advantages of parametric surfaces. The PDE method enables us to generate facial animation from a given complex face model by adjusting only a small set of boundary curves. This methodology enable us to produce a set of template mesh surfaces that are used to transfer a given motion sequence to a given human head mesh model. This section will explain in more detail the implementation of the talking head environment.

6.5.1. Generation of viseme poses

Viseme-driven speech animation approaches often require manual design of key mouth poses in order to generate realistic speech animations. The first step for building the facial animation system consists of generating the facial expression data [103]. The template face in Figure 6.6 (a), is represented as a set of 28 boundary curves extracted from a laser-scanned 3D face model in its neutral configuration. The curve-set covers the entire face area, describing the most important facial features. The PDE face surface is then reconstructed using a combination of nine different fourth order PDE that guarantee surface continuity is shown in Figure 6.6 (b). Note that the resolution of the parametric domain of PDE surface is set to 35x35. As mentioned before, a set of pre-configured expressions is required for animating speech [136]. A database of 22 visemes is used to identify the corresponding viseme of a given word or

phrase within the TTS engine. The talking head database consists of 15 viseme expressions; some viseme can be repeated to save loading time e.g. viseme AW and OY can be represented using the same viseme.

(a) (b)

Figure 6. 6. (a)The neutral pose template curve-set. (b) The resulting surface.

Each viseme is represented as a curve-set derived from a face mesh model as seen in Figure 6.6 (b). The viseme curve-set poses have been obtained from the work presented in [137]. The authors have developed a PDE-based facial deformation technique that uses the boundary curves of a PDE surface to deform a face model using the MPEG-4 compliant facial feature points. The idea behind this approach is that it utilizes the boundary curves for complex facial deformations rather than using conventional control points and surface interpolation techniques. A group of boundary points anatomically related to facial features, such as right corner of left eyebrow, left corner of inner lip contour, are selected as feature points according to the MPEG-4 definition. Animation of the face model is achieved by adjusting the position of the boundary curves before each calculation of the PDE surface.

The deformations have been computed using a series of weighted sinusoids to parameterize the FAP-driven facial animation with the feature points. Additional

boundary points have also been selected and used as weights in areas around the feature points in order to guarantee surface smoothness. The mouth and eyes area are achieved using sinusoidal animation of the corresponding feature points. For stretching the lip corners, linear interpolation is applied to the boundary curve points representing the lips.

The resulting curve-sets are stored in an internal viseme database and they will be used for computing a PDE surface that passes through the control points of each curve to generate a surface representation. The process is shown in Figure 6.7. Each curve-set viseme is used as the boundary conditions required for calculating the PDE method. The new surface is stored and used later for retargeting the facial deformations of the corresponding viseme to a different face mesh model.

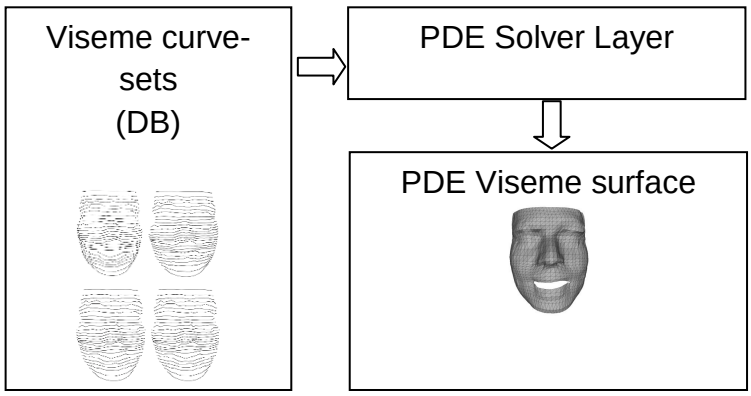


Figure 6. 7. PDE–based viseme process.

The preconfigured viseme curve-sets contain all the necessary information to generate the PDE-based speech expressions for the animation system. Storing only a set of curves rather than a mesh object for each required viseme pose gives us the advantage at keeping the storage requirements at minimum. This technology can be exploited to reduce network transmission bit-rates, by

sending only animation parameters rather than the video sequence. Figure 6.8 contains various template viseme poses for calculating the required PDE template expressions. The curve sets seen in Figure 6.8 (top), are used to generate the human face mesh for the given viseme and apply motion retargeting to transfer the deformations to a different target human face model. Visemes *A, B/M/P*, *EE* and *F/V* used for the speech animation are shown in both configurations, curve and mesh representation.

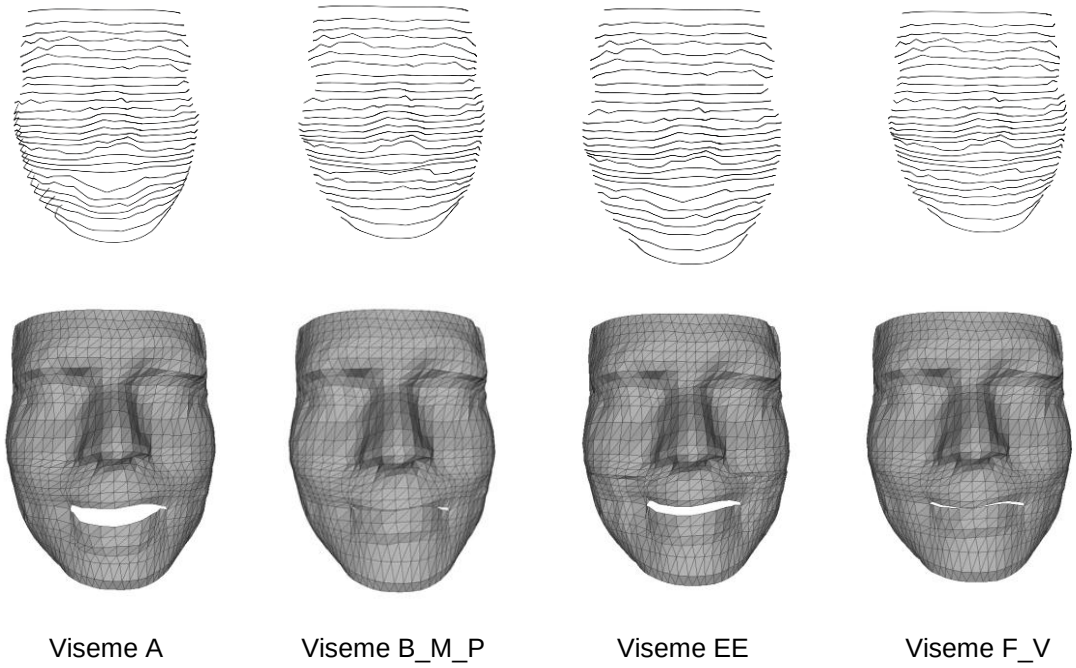


Figure 6. 8. Curve based viseme poses (Top). Template surface for corresponding curve-sets (Bottom).

6.5.1.1 Blend shape visemes

After the curve-sets have been computed using the PDE method, the generated PDE surfaces for each viseme in the database are stored and represented in the system as a group of polygon meshes. This is a pre-processed procedure that takes place at the loading time of the application and it is repeated for all the required visemes in the database. The next step consists of generating

animation for any two given facial expressions. The talking head system must be able to blend any required viseme poses in real time. For this task shape interpolation between two key-frames is used, where two or more facial expressions are captured and in between frames are computed by linear interpolation.

Linear interpolation is used to approximate a value using two known values of that function at other points. Speech animation of two given facial expressions is produced using the code in Equation 2 below. The new position is calculated by interpolating the two given visemes , pos_1 and pos_2 ; these indexes are computed from the text-to-speech engine and assinged a value according to the viseme mapping in the database. The interpolation is calculated for all the vertices in the x, y, z coordinates of each viseme. The time variable, represented by t is computed using the current frame of the animation and is used to approximate the new position at time t . Additionally, the normals of the new surface are interpolated using the same techinque.

$$t = (\text{Duration} - \text{CurrentFrame}) / \text{Duration}.$$

$$\text{newMesh} = t * \text{Mesh}[\, pos_1 \,] + ((1 - t) * \text{Mesh}[\, pos_2 \,]), \qquad (6.2)$$

The talking head system presented in this work integrates emotions that are produced from the AIML chatterbox engine according to the current user input. This requires the shape interpolation of an additional expression of the overall speech animation. In this case, the additional expression is included in the

interpolation function with an additional variable to control the amount or intensity of the blend shape. The intensity variable is passed to the interpolation in real time during the calculation to produce a realistic conversation during the speech animation.

The process of speech animation is explained in the diagram shown in Figure 6.9 and shows the communication between the various layers of the speech animation. Animation between two viseme poses is dependent on the user input. Input text is send to the chatterbot process to generate a response; response from the AIML engine is then processed by the TTS engine to produce the appropriate speech and viseme information used for the animation. The corresponding visemes will be parsed to the Blend Shape process to produce the interpolated shape at any given time. Additionally, a third expression is parsed to the shape interpolator indicating the mood change. This information is obtained from the AIML engine in real time. This process will be explained later in this chapter.

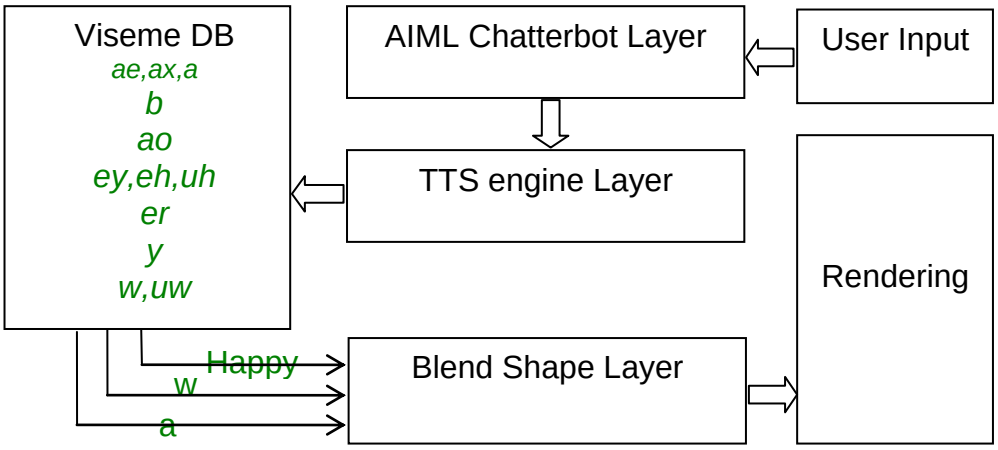


Figure 6. 9. The speech animation process layers.

6.5.2 Text-to-speech engine integration.

As discussed in Section 6.2, the TTS engine integration is handled from the Microsoft SAPI 5.0 API using the .NET framework. The SAPI provides all the necessary functionality for controlling various speech related properties in real time. The diagram in Figure 6.9 shows the communication between the TTS engine and the Viseme database. The output of the Rebecca chatterbot is captured and synthesized by the TTS engine. Events are used to synchronize the output speech and to produce visemes from input text. Each viseme is linked using the SAPI specified **SPEVENTENUM** list and the talking head viseme database, (Table 6.1 Section 6.2), and then passed to the blend shape layer.

6.5.3 AIML integration

The next implementation consists of integrating the AIML based chatterbot to the talking head system. This process can be thought as an additional layer dependant to the user's input. The current speech animation system has been tested with two different AIML interpreters, the Rebecca AIML and the AIMLBot, which is only compatible with the .NET framework. Both interpreters are satisfactory for the level of use in this application. The latest version of the Talking head system developed here uses the AIMLBot because of its simplicity to operate. The results of the AI process are not affected since all the required data are contained in the AIML files that are common to both interpreters. Initializing the AIML engine requires loading the appropriate AIML files that will be used for the conversation. These files, obtained from the A.L.I.C.E bot repository, are a set of XML based scripts that contain A.L.I.C.E's brain.

Additional XML files are also required to set up various bot personalization attributes such as name, interests, favourite phrases etc.

One of the many advantages of using AIML is that it is fully customizable. The bot master can include new AIML content with custom handling of input text. As discussed in Section 6.3, implementing custom AIML elements can add intelligence to the bot and make the human-computer interaction more believable, accurate and human like. Additionally, an infinite number of new tags can be added to extend the functionality of the bot. The selection of current mood in the talking head system is an example of AIML customization. The emotions have been embedded in the AIML files by introducing a new tag called <emotion> and a “name” property (19). The property name will hold the mood name of the current AIML pattern element. This information has been added manually in a custom AIML file to support a range of different input pattern elements. Figure 6.10 below, shows an example of emotion handling in AIML. When the input text matches the pattern element, the template element contains additional information. In this example, the response is produced by a <random> tag that contains two different responses and the emotion that is set for current template is happy. However, the new <emotion> tag requires additional processing since the AIMLBot interpreter does not recognize custom tags. A simple XML node processing function searches the <category> node element for each new response to check if it contains a valid emotion element. If an emotion element is found, it is parsed to the Blend Shape layer to produce the new expression. Alternatively, if the current template does not contain any emotion element, then the current mood is set to neutral.

```
<pattern>I LIKE YOU</pattern>
  <template>
   <emotion name="happy" />
    <random>
    <li>  I Like you too.</li>
    <li>  You are very kind. Thank you!.</li>
   </random>
</template>
```

Figure 6. 10. An AIML example containing emotion data.

6.6 Motion retarget

Finally, once the system is initialized with the PDE-based viseme expressions and synchronized with the AIML engine, it is required to retarget [138] the deformations to a different target face mesh model. More information about the motion retargeting technique used in this work can be found in chapter 5.5. This procedure is repeated for every expression in the viseme database. There are cases where a better mapping correspondence is required, e.g. mouth regions between the two objects, to achieve a more detailed representation. To solve the problem, the template face and the target object are split into face region maps, such as bottom and top mouth, nose and eyes. Each map contains the index of each vertex that is included in that region. The collection of these points is handled using the Autodesk Maya environment, where the selection and output of the correspondent vertices is performed using MEL scripts. The correct selection of these points is very important since the motion transfer is based on the vertex indexes each map contains.

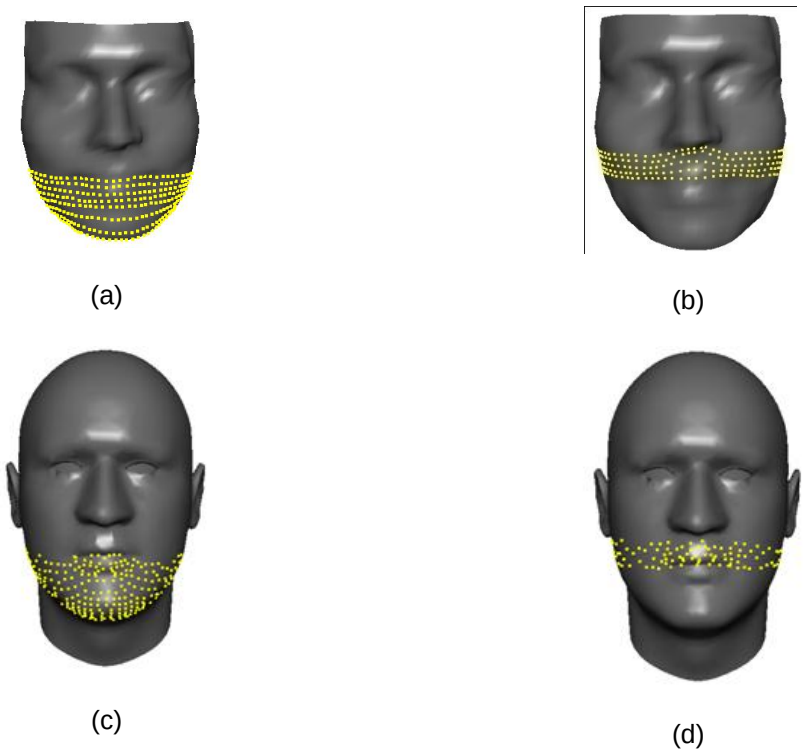


Figure 6. 11. Region maps used for motion retarget.
template face mouth region maps (top). Target mesh mouth region maps (bottom).

Figure 6.11, contains the top and bottom mouth region maps for the template face and the target human face object. Motion retarget is applied separately to each region map until all the required facial areas are processed. Once the motion transfer is complete, the resulting object can be associated with the expression it represents in the viseme database. Note that the quality of the mapping correspondence between the template viseme and the output face model depends also on the resolution of the grid used to compute template surface representation. Thus, using a grid with a similar number of points to the number of vertices in the original mesh will produce a better mapping correspondence. It can be seen in the diagram in shown Figure 6.12, that the motion retargeting process can be visualized as an additional layer in the talking head system that communicates between the PDE surface layer and the input mesh model.

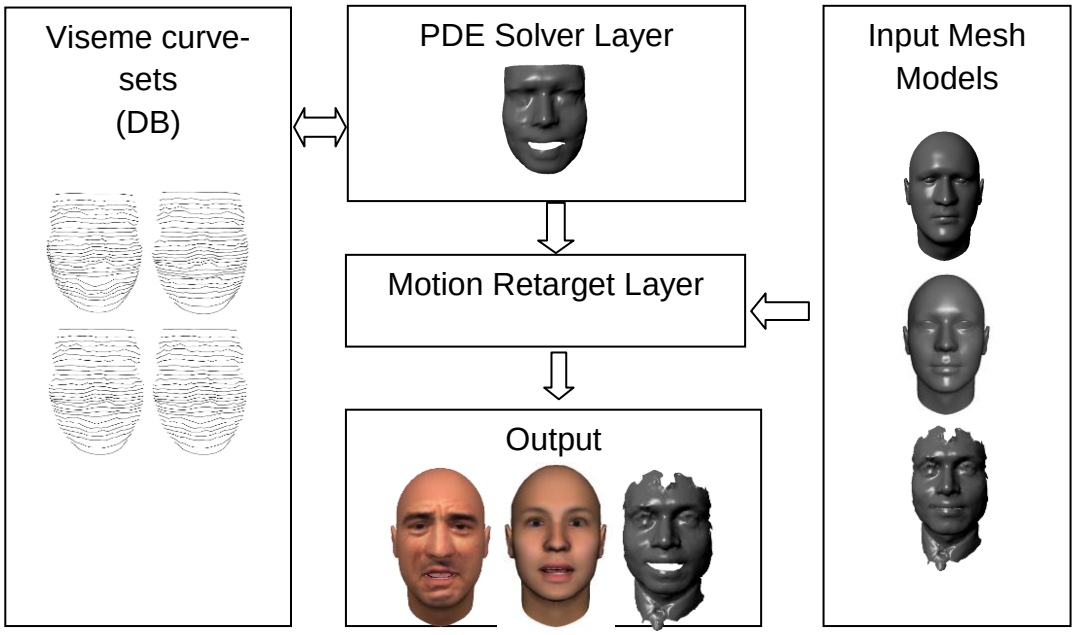


Figure 6. 12. Motion retarget process.

6.7 Examples

Figure 6.13, contains a sequence of viseme expressions that are transferred from the template mesh (a, b, c) to the different target face mesh model (d, e, f). The motion retargeting technique employed in this work requires a mapping correspondence between the two objects, such as each point of the target mesh model is associated with the nearest point of the template surface. This way each point of the target mesh will be represented on the template model. Finally, motion retargeting is carried out by adding the difference between each point in the source model and the corresponding point in the target model.

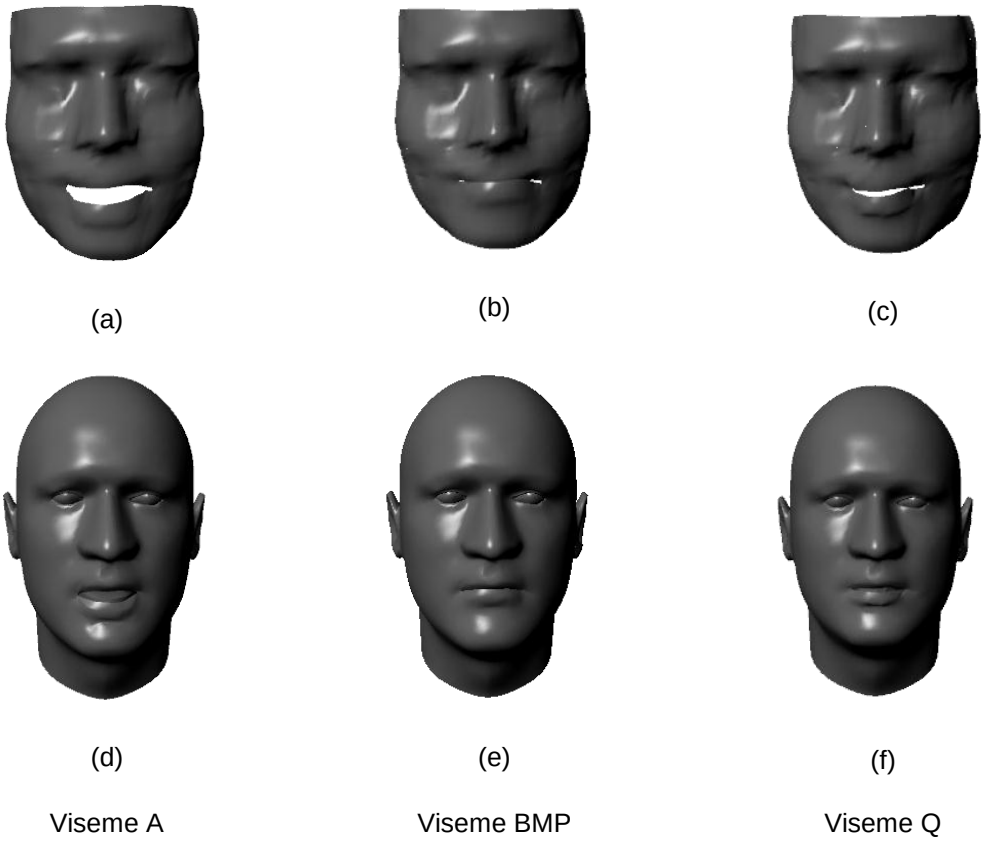


Figure 6. 13. Viseme expressions. PDE-based viseme expressions. (a,b,c) Motion retargeting to human head object. (d,e,f)

Next the sequence of expressions in Figure 6.14 shows several examples of motion retargeting on expressions that are used within the talking head system to simulate change of mood during the speech. These expressions are included in the blend shape process to adjust the current viseme according to the current mood expression. The current Mood selection can be controlled by certain input text, duration or from the AIML engine [133]. AIML elements can encapsulate the response and certain mood that is generated from each parsed input text. This way, the response from the bot can also contain the appropriate mood expression.

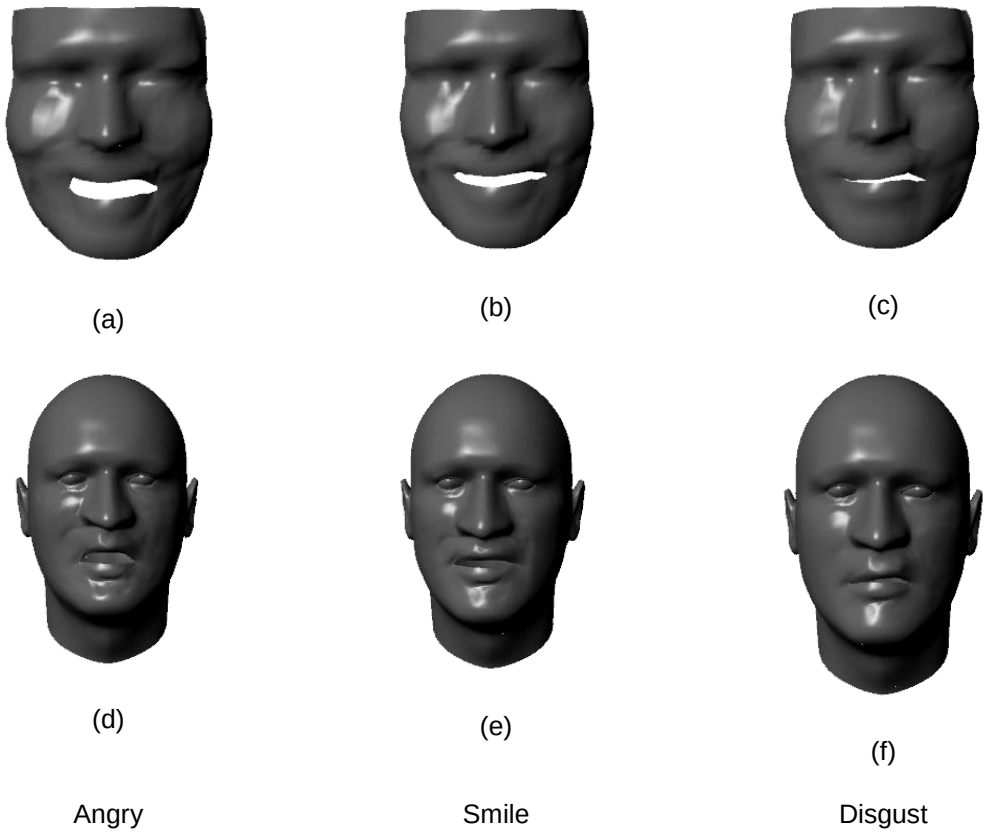


Figure 6. 14. Emotion expressions. PDE-based emotion expressions. (a, b, c) Motion retarget to human head object . (d, e, f)

Another example of motion retargeting is shown in Figure 6.15; the target 3D human face has been replaced with a model acquired from a 3D scanner. The initial alignment of the two models plays a very important role in the correct motion transferring between the facial region maps. There are cases where facial regions of the target mesh need to be removed from the motion retarget process. The generic face template used in this work contains 4 basic regions maps; for more realistic facial deformations the number of face area maps must be increased according to the facial features of the target objects.

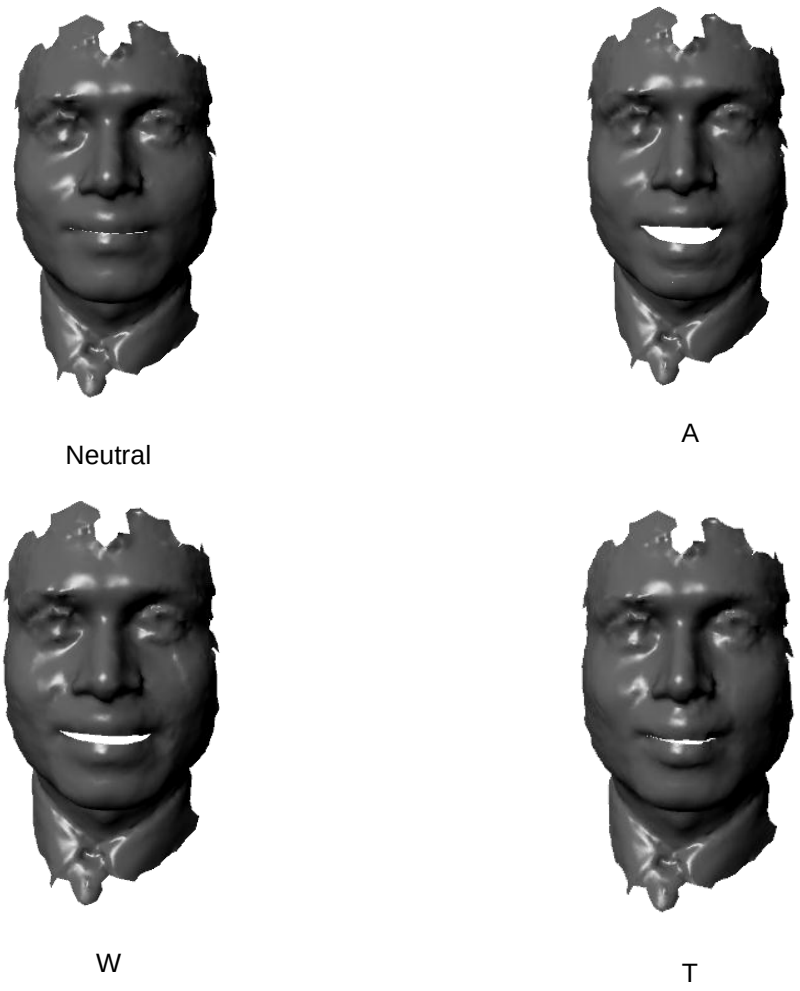


Figure 6. 15. Motion retarget to a 3D scan mesh model.

The final textured version of the animated target object is shown in Figure (6.16 -6.17). The animated face contains teeth, tongue and mouth to increase the realism of the speech animation. The position of the teeth and tongue is interpolated alongside with the viseme blend shape process.

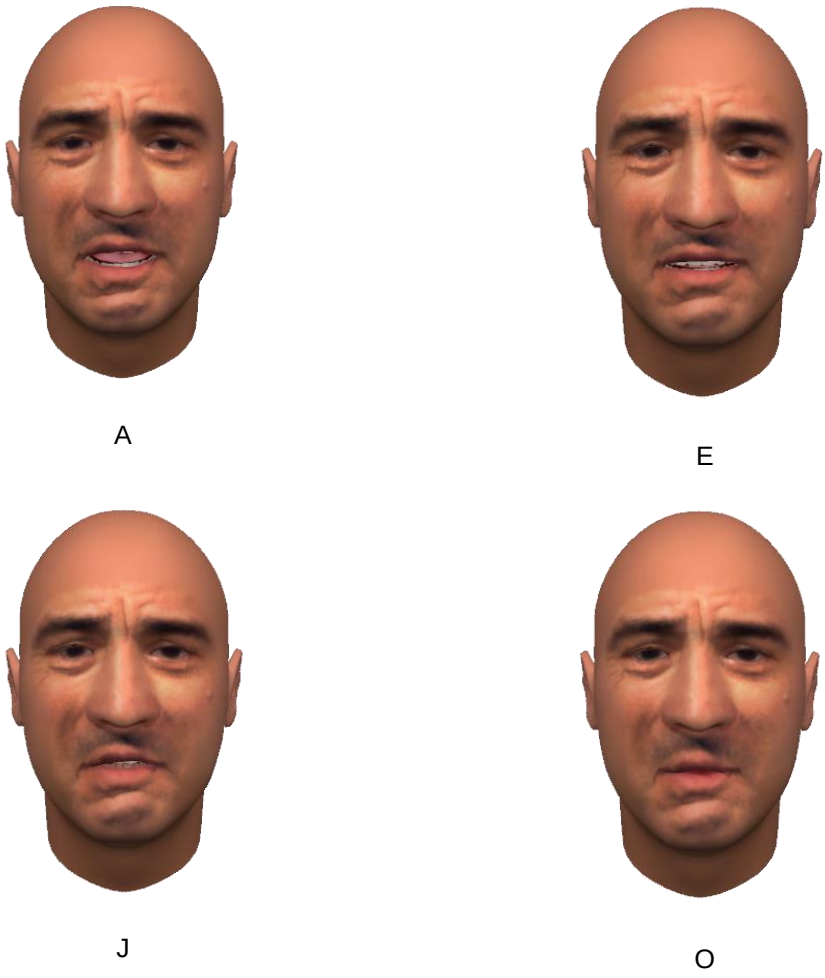


Figure 6. 16. Final rendering of the talking head system 3D face. The face model contains mouth, teeth and tongue animated according to each viseme.

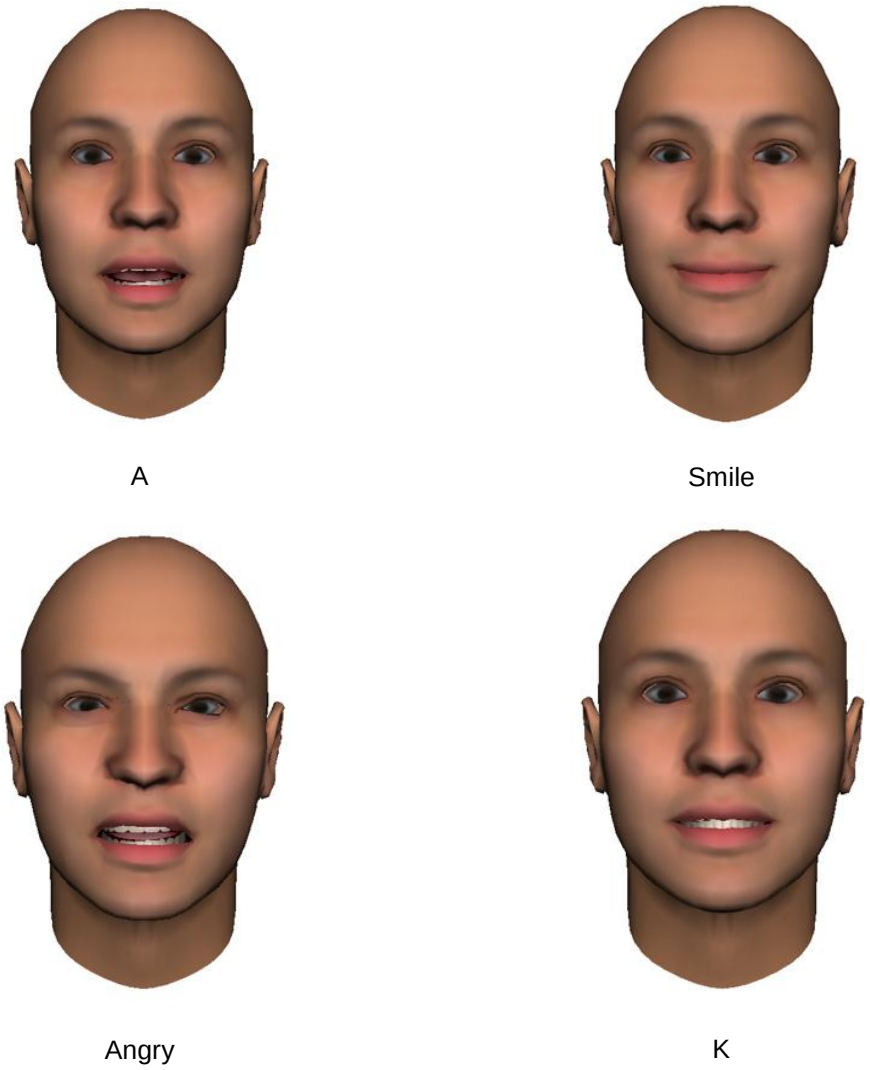


Figure 6. 17. Final rendering of the different face model retargeted to the talking head system.

6.8 Conclusions

This work presents a technique for animating PDE-based facial expressions within a talking head system. It uses a set of pre-configured curves representing various viseme poses to calculate a template PDE surface. The resulting surface is used to associate various facial feature areas with a different target face mesh model. Motion retarget is then applied to transfer the deformations in these areas from the template to the target model. This technique offers animation re-usage, since all the necessary viseme poses for the animation are pre-calculated and re-used for a different target mesh model. Additionally, it minimizes the storage requirements by storing only a small set of curves for each expression. The system interacts with the user using an AIML chatterbot to generate response from input text.

The user can enter a question or phrase and the AI bot will generate the appropriate answer to facilitate a real time conversation. The response is then captured and converted to speech from the text-to-speech engine and each word is split into a sequence of viseme poses that are used to synchronize the facial animation. The animation is carried out by linearly interpolating a given set of visemes to generate the in between transition of different visemes. An improvement to be included in this technique is a more generic template PDE model representation that could enable the animation to a larger variety of facial models. Future work can be undertaken in automating the facial map extraction process. This process is required to associate various facial areas between the two face models during the motion retargeting process.

Chapter 7: Conclusions and future work

In this work several techniques were presented that exploit parametric surfaces obtained through the use of the PDE method. The PDE method has been used as the surface generation technique since it offers many advantages over traditional surface generation techniques. It has been shown in previous chapters that it can reduce dramatically the amount of information required to represent complex three-dimensional surfaces. This is due to the small number of design parameters that are required to describe a surface. Moreover, the PDE method offers several advantages over the design, manipulation and animation of a surface, some of them have been adapted in a series of methodologies presented in the previous chapters of this thesis.

During the writing of this thesis, many goals where achieved. The implementation of the techniques presented in the previous chapters helped to give a better and clear understanding of the PDE method, its characteristics, advantages and disadvantages. However, the need for developing various applications gave us the opportunity to explore several programming languages and open source libraries in order to produce virtual environments capable of interacting with the PDE method. Throughout the research and development that took place over the last few years, several techniques that are not covered in this thesis were also adapted for developing efficient software designs. Since the implementation of the above methodologies is software based, the knowledge and understanding of good software development practices are necessary in order to design implement and maintain virtual environments.

Moreover, some of these techniques have more to offer than others. Particularly, the conversion of a mesh model to a patch wise parametric surface is one of these techniques which aim is to solve a very difficult problem.

Currently, many methodologies are trying to solve the problem of mesh parameterization using a variety of approaches; so far the results shown that there is still room for introducing a better more flexible technique that will be able to support any given mesh model and at the same time produce flexible compressed data. Doing so it will increase the use of parametric surfaces in a wider perspective, making possible to integrate them in systems where the storage and computational requirements are required to be minimum. Such systems including vast gaming environments and interactive cyber worlds will benefit from the usage of PDE surfaces by utilizing all of their characteristics in such areas as geometry, animation and physics.

7.1 Achievements

The main contribution of this thesis has been achieved through the adaptation of the PDE method in various applications focusing on modelling, manipulation and animation of PDE-based surfaces. One of the main challenges in computing a PDE surface lies in generating the set of boundary conditions that represent the outer contour of the surface in question. The steps involved in this procedure have been identified and presented in Chapter 3. A new technique was developed that automatically describes a surface in terms of boundary curves. The process starts by reducing an input mesh model using a mesh simplification technique until it reaches a satisfactory level of quality and number of faces. Each face of the reduced surface is then converted to a boundary patch consisting of a fixed number of control points. The curvature and smoothness of the original mesh surface is extracted using a ray casting technique between each of these new control points and the original mesh surface before the mesh simplification. This process is applied in every face of

the simplified model in order to identify and extract all the feature points that describe the input mesh surface.

Some of the problems identified during this process, consists mainly on the complexity of the model in question. To that extend, cases where the model consists of sharp edges, non-manifold geometry and complex uv-mapping need to be identified and handled in different manner before and after the simplification process. For example, during the mesh reduction process, it might be necessary to keep some sharp edges in order to maintain special features on the original model. In such case, the triangular face that contains shard edge needs to be handled separately when extracting the patches. However, in order to preserve a sharp edge, an additional set of faces need to be identified and preserved during the mesh simplification leading to storing and converting into PDE-patches additional number of faces. Moreover, the current technique is capable of producing acceptable results in models with smooth topological areas. The aim of this technique can be summarized as a solution to the problem of mesh segmentation for converting a given mesh model to set of parametric PDE-based patches; this is done by utilizing a process that automatically identifies and produces boundary-based patches that represents the given model's topology.

Various implementations of the PDE method in various environments utilizing scripting language were presented in Chapter 4. Some of these include the development of an interactive environment for designing and manipulating PDE-based aircraft configurations and the integration of the PDE method in a CAD environment, such as Maya. The first technique provides the user with tools to simplify the design of aircraft configurations using parametric PDE surfaces in a stand-alone 3D environment. Whereas, the manipulation of such

configurations is achieved by simple affine transformations that are applied to the boundary conditions of the PDE method. The second methodology of Chapter 4 presents a plug-in for the Maya environment that makes possible the calculation of PDE surfaces for a given set of boundary conditions. The advantage of such approach can be mainly seen as the incorporation of various tools that exist in the Maya environment in the construction of PDE surfaces. Additionally, this work has proven useful in expanding the PDE method capabilities by introducing a new surface deformation technique, while making possible the construction and manipulation of PDE surfaces within a CAD environment.

Animation is another area in computer graphics were the PDE method has been successfully adapted. Several techniques were developed to show the advantages of such surfaces by utilizing existing animation approaches as well as presenting new ones. Chapter 5 presents an animation technique based on the PDE method that generates human-based cyclic motions like walking and running. It makes use of simple mathematical expressions to generate cyclic motions. Periodic functions, such as sines and cosines, are used to produce the general motion of the human body, which succeed in making the movement very realistic. These expressions are then attached to a skeleton system that holds the boundary conditions that represent a three-dimensional human character. An advantage of this technique is that it can produce realistic human based cyclic animation of a PDE-based surface and retarget the motion back to the original mesh model. One of its limitations is that the character model that needs to be animated must meet the topological requirements of the PDE-based template representation. For example, non-human character models cannot be processed correctly since they might contain additional body

parts that will not fit the template character used for the animation. As an extension of this technique, a better template representation can be developed that can either support a wider selection of models or ideally will be able to adapt to any given surface.

A second technique presents a different way of animating PDE-based surfaces by utilizing the spine of the surface. This methodology has been developed to simulate fish locomotion using the spine of a PDE surface representation of a given fish. The spine associated with PDE-based fish model is manipulated analytically through the use of a set of parameters that are capable of producing various fish movements. The advantage of this approach is that it can generate complex animation without the need of an additional skeleton hierarchy, all the information required to animate or deform a model are contained in the shape of the surface itself. This methodology contains the same limitations with the cyclic animation of human character technique; this is due to the fact that both techniques are using a template representation for transferring the animation between the PDE geometry and the original model.

The last methodology presented in Chapter 6 is using PDE-based surfaces for generating viseme-based facial animation. This approach consists of a dynamic talking head system that can animate a database of three-dimensional face expressions by utilizing the PDE method. This technique uses a set of pre-configured template boundary conditions representing all the required viseme poses for the simulation of a human-computer based interaction system. Initially every template viseme pose is evaluated using the PDE method, and then used to transfer the PDE-based pose to a given face mesh model by applying motion retargeting. This technique, offers animation re-usage, since all the necessary viseme poses for the animation are pre-

calculated and re-used for a different target mesh model. Additionally, the system offers a computer-human interaction through the use of an AIML chatterbot to generate response from input text. The user can enter a question or phrase and the AI bot will generate the appropriate answer to facilitate a real time conversation. The response is then captured and converted to speech from the text-to-speech engine and each word is split into a sequence of viseme poses that are used to synchronize the facial animation. This process can automate the animation of different characters in a computer-human interaction environment by re-using the same pre-build expression and viseme set.

7.2 Future work

A list of improvements can be identified for each methodology that is presented in this thesis. Each approach can be either expanded to include several additional features or improve its functionality. Starting with the automatic curve extraction technique, future enhancements can be undertaken in handling a wider variety of geometrical topologies, where problems such as maintaining special features need to be indentified and then converted into a PDE-based patch or kept as triangular faces. Different methodologies need to be developed in order to maintain features like texture coordinates in the generated PDE-based patches. However, the segmentation of the polygon mesh using a reduced version cannot maintain such features. To that extend, new techniques for generating boundary patches while maintaining the original surface curvature and texture coordinates need to be implemented. Such techniques include the generation of boundary regions using the uv mapping, curvature and the normal crease angle of a given surface.

The motion retargeting technique that was introduced and utilized in various applications throughout this thesis can also be improved in various ways. One of the main disadvantages of such technique is that it is not fully automated; generating the facial area maps for re-using the facial animation of the talking head system required the manual selection of these facial points using Maya. Additionally, the process might produce various deformations while retargeting the motion to a topologically different target model. Although, the particular motion retargeting technique is application specific; it was originally developed to transfer motion between a PDE representation and a polygon model with similar topology, there is still room for improvement. Some of the future enhancements can be identified as adding constraints at geometric areas that cannot be successfully transferred and generate automatically area maps by identifying various topological surface features.

Additional future work can be undertaken on improving the interactive aircraft designer environment. A variety of aircraft parts and transformation tools for enabling complex designing of airplanes can be included to improve the overall appearance of the final surface; while adding export functionality to various formats for use with commercial CAD packages can be proven useful. Future work could also be undertaken in determining the stress and strain in materials and structures subjected to static or dynamic forces that are applied to an aircraft configuration during the flight.

In the field of animation, the PDE method was proven useful since it provides tools for parameterizing and designing cyclic motion. The use of the PDE spine as a universal rigging tool for PDE surfaces can be seen as another future improvement that can unify the work presented in Chapter 5. Once a PDE representation is obtained over a given mesh model, the spine of the

surface can be identified and analytically controlled in order to create specific cyclic motions. As an example, the animation of the PDE-based human representation shown in Chapter 5 was animated using a skeleton hierarchy. An alternative to that technique will be to identify the PDE spine for each body part and set various constrains in places where the joints are required to move for creating human based motion. For each part, a different parameterization and an analytic solution need to be produced in order to generate the animation. An advantage of such technique is that no additional data for producing the skeleton hierarchy are necessary.

Finally, future improvements can be undertaken in enhancing the functionality of the PDE-based talking head system. Some of them include, publishing of the application in a web page since the data have been optimized for internet transfer, including a wide variety of emotion expressions and a real-time mood system. Such features can enhance the overall simulation and create a very realistic human-computer interaction environment. Moreover, implementing a skeleton system to simulate head movement according to a cyclic function or the response from the AI can increase the realism of the conversation.

Besides the various future improvements that can be undertaken in order to improve several aspects of each methodology, there are many technologies involved that were necessary in order to make the evaluation and visualization of a PDE surface possible. For example, during the construction of the boundary curves representing the human character used for cyclic animation, the boundary data were extracted manually from Maya. This process requires the creation and manipulation of parametric curves using tools embedded in the Maya environment. However, once the data were created, the communication

with the PDE method libraries and the skeleton hierarchy would be possible only by using a scripting language such as MEL script. To that extend, various scripts were developed to extend the capabilities of the software and provide a better user interaction while evaluating a PDE surface. Alternatively, the rendering of parametric PDE surfaces required the use a Rendering API specification library. The implementation of rendering functionality using the OpenGL specification made possible the drawing of materials, normals and texture coordinates while enabling the selection of components such as curve, faces and objects from the user. During the implementation and integration of such functionalities, many problems occurred. Some of them were due to the complexity of the problem to solve, while others were produced due to library specific problem, hardware compatibility and software limitations. The PDE method is an approximation technique that aims to smooth the data during its evaluation; this is identified as one of its main disadvantages since representing non-smooth surfaces appears to be a challenge. Nonetheless, each application contains several features that can overcome any problem that might appear during its execution process and can provide a novel functionality for interacting with PDE surfaces.

To conclude, several applications were presented in this thesis utilizing various methodologies but with a shared objective, to exploit the PDE method and show the advantages over traditional surface generation methods. Further work can be undertaken to produce better and more detailed results depending to the end user application. Moreover, depending on the complexity of the mesh model in question, the techniques that have been developed need to be adapted in order to take into account various topological criteria that define a surface. However, the surface generation technique utilized here contains

several important features that vary from the ability to reduce the data required to characterise a surface to the availability of tools for deformation, manipulation and animation of a surface without additional tools or constraints.

Bibliography

1. H.Bartels, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. 1987, USA: Morgan Kaufmann.
2. M.Hazewinkel, *Encyclopaedia of Mathematics, Supplement III.* 2002: p. 568
3. B.Fornberg and J.Zuev, *The Runge phenomenon and spatially variable shape parameters in RBF interpolation.* Computers & Mathematics with Applications, 2007. **54**(3): p. 379-398
4. L.Piegl and W.Tiller, *The NURBS Book*. Vol. 2nd edition. 1997: Springer.
5. L.Piegl, *On NURBS: a Survey.* IEEE Computer Graphics and Applications, February 1991. **11**.
6. K.S.Fu and T.L.Booth, *Grammatical inference: Introduction and survey: Part i & ii.* IEEE Transactions on Pattern Analysis and Machine Intelligence., 1986. **8**(3): p. 343-376.
7. J.Bloomenthal, *Implicit Surfaces.* Encyclopedia of Computer Science and Technology Marcel Dekker Inc., NY, 2000.
8. A.Knoll, *A Survey of Implicit Surface Rendering Methods, and a Proposal for a Common Sampling Framework.* Visualization of Large and Unstructured Data Sets 2008: p. 164-177.
9. OpenGL. *http://www.opengl.org/documentation/*.
10. J.Bloomenthal, *Polygonization of Implicit Surfaces.* Computer Aided Geometric Design, Elsevier, 1988.
11. J.C.Hart, *Ray Tracing Implicit Surfaces.* Course Notes: Design, Visualization and Animation of Implicit Surfaces SIGGRAPH 1993.

12. B.Wyvill, C.Wyvill, and J.Bloomenthal., *Implicit Surfaces for Geometric Modeling and Computer Graphics.* Course notes SIGGRAPH ACM., 1996.

13. B.Wyvill and K.Overveldz., *Polygonization of Implicit Surfaces with Constructive Solid Geometry.* International Journal of Shape., 1996.

14. L.Schumaker, *Spline Functions: Basic Theory*. Cambridge Mathematical Library Vol. 3rd. 2007: Cambridge University Press.

15. M.Bertalmio, et al., *Variational Problems and Partial Differential Equations on Implicit Surfaces: Bye Bye Triangulated Surfaces? .* Geometric Level Set Methods in Imaging, Vision, and Graphics., 2003. **VII**: p. 381-397.

16. M.Bertalmio, et al., *Image Inpainting.* Siggraph ACM., 2000: p. 417-424.

17. M.I.G.Bloor and M.J.Wilson, *Generating Blend Surfaces using Partial Differential Equations.* Computer Aided Design 1989. **21**(3): p. 165-171.

18. H.Ugail, *Parametric Design and Optimisation of Thin- Walled Structure for Food Packaging.* Journal of Optimisation and Engineering, 2003. **4**: p. 291-307.

19. H.Du. and H.Qin, *Direct Manipulation and Interactive Sculpting of PDE Surfaces.* Computer Graphics Forum, 2000. **19**(3): p. 261-270.

20. M.I.G.Bloor and M.J.Wilson, *Spectral Approximations to PDE Surfaces.* Computer Aided Design 1996. **82**(2): p. 145-152.

21. H.Ugail, M. I. G.Bloor, and M. J.Wilson, *Techniques for Interactive Design Using the PDE Method* ACM Transactions on Graphics, 1999. **18** (2): p. 195-212.

22. WE.Williams, *Partial Differential Equations*. 1980, New York.: Oxford University Press.

23. G.D.Smith., *Numerical Solutions of Partial Differential Equation.* 1987, New York: Oxford University Press, .

24. H.Ugail, *Parametric Surface Meshing for Design Optimisation using a PDE formulation.* Numerical Grid Generation in Computational Field Simulations, 2002: p. 715-724.

25. G. González Castro, et al., *A Survey of partial differential equations in geometric design.* The Visual Computer, 2008. **24**(3): p. 213-225.

26. H.Ugail, M.I.G.Bloor, and M.J.Wilson, *Manipulation of PDE surfaces using an interactive defined paramerization.* 1999.

27. H.Ugail, *Method of Trimming PDE Surfaces.* Computers & Graphics, 2006. **20**(2): p. 225-232.

28. G.Xu, Q.Pan, and C.L.Bajaj., *Discrete surface modelling using Partial Differential Equations.* Computer Aided Geometric Design, 2006. **23**: p. 125-145

29. H.Ugail, *On the Spine of a PDE Surface.* Lecture Notes in Computer Science, 2003. **2768/2003**: p. 366-376.

30. M.Athanasopoulos, G.González Castro, and H.Ugail., *Cyclic Animation of Human Body Using PDE Surfaces and Maya.* International Conference on CyberWorlds., 2009.

31. M.Athanasopoulos, H.Ugail, and G. González Castro, *Parametric design of aircraft geometry using partial differential equations.* Elsevier Science Ltd., 2009. **40**(7): p. 479-486.

32. M.Attene, S.Katz, and M.Mortara, *Mesh segmentation – A comparative study.* IEEE International Conference on Shape Modeling and Applications, 2006: p. 7.

33. A.Agathos, I. Pratikakis, and S.Perantonis, *3D Mesh Segmentation Methodologies for CAD applications.* Computer-Aided Design & Applications, 2007. **4**(6): p. 827-841.

34. M.Vieira and K.Shimada, *Surface mesh segmentation and smooth surface extraction through region growing.* Computer Aided Geometric Design, 2005. **22**(8): p. 771-792

35. N.S.Sapidis and P.J.Besl, *Direct construction of polynomial surfaces from dense range images through region growing.* ACM Transactions on Graphics 1995. **14**(2): p. 171 - 200.

36. M.Attene, B.Falcidieno, and M.Spagnuolo, *Hierarchical mesh segmentation based on fitting primitives* Visual Computer, 2006. **22**(3): p. 181-193.

37. S.Katz and A.Tal, *Hierarchical Mesh Decomposition Using Fuzzy Clustering and Cuts.* Siggraph ACM, 2003: p. 954 - 961.

38. Y.Lee, S.Lee, and A.Shamir., *Intelligent Mesh Scissoring Using 3D Snakes.* Proceedings of the Computer Graphics and Applications, 2004: p. 279 - 287

39. Y.Lee, S.Lee, and A.Shamir., *Mesh scissoring with minima rule and part salience.* Computer Aided Geometric Design., 2005. **22**(5): p. 444 - 465.

40. W.Kiefer, *Intelligent Scissoring for Interactive Segmentation of 3D Meshes.* 2004, Princeton University.

41. D.H.Kim, I. D.Yun, and S.U.Lee, *Triangular Mesh Segmentation Based On Surface Normal.* 5th Asian Conference on Computer Vision, Australia., 2002.

42. J.O.Talton, *A Short Survey of Mesh Simplification Algorithms* 2004, University of Illinois at Urbana-Champaign.

43. M.Garland., *Quadric-Based Polygonal Surface Simplification.*, in *School of Computer Science.* 1999, Carnegie Mellon University. : Pittsburgh,USA. p. 200

44. W.J.Schroeder, J. A.Zarge, and W. E.Lorensen, *Decimation of Triangle Meshes William.* ACM SIGGRAPH, 1992: p. 65 - 70

45. T.Wolle, H. L.Bodlaender, and T.Wolle, *A Note on Edge Contraction* 2004, Department of Information and Computing Sciences, Utrecht University.

46. J.Bloomenthal and K.Ferguson, *Polygonization of Non-Manifold Implicit Surfaces.* SIGGRAPH, 1995.

47. H.Hoppe, T.DeRose , and T.Duchamp, *Mesh optimization.* Proceedings of the 20th annual conference on Computer graphics and interactive techniques, 1993: p. 19 - 26.

48. M.Garland and P.S.Heckbert, *Surface Simplification Using Quadric Error Metrics.* SIGGRAPH, 1997: p. 209 - 216.

49. E.Haines, *Essential ray tracing algorithms*. 1989, London, UK: Academic Press Ltd. 33 - 77.

50. A.Glassner, *Space Subdivision for Fast Ray Tracing.* IEEE Computer Graphics & Applications,, 1984. **4**(10): p. 15-22.

51. J.Amanatides and A.Woo., *A Fast Voxel Traversal Algorithm for Ray Tracing.* Eurographics, 1987: p. 3-10.

52. A.Fujimoto and K.Iwata, *Accelerated Ray Tracing.* IEEE Computer Graphics and Applications, 1986. **6**(4): p. 15-26.

53. N.E.Sevant, M.I.G.Bloor, and M.J.Wilson, *Aerodynamic design of a flying wing using response surface methodology.* AIAA J Aircraft., 2000. **37**(4).

54. N.E.Sevant , M.I.GBloor , and M.J.Wilson, *Cost effective multipoint design of a blended HSCT.* AIAA J Aircraft, 1998. **36**(2): p. 642-50.

55. M.I.G.Bloor and M.J.Wilson, *Efficient parameterization of aircraft geometry.* 32, 1995. **Journal Aircraft**: p. 1269-1275.

56. H.Sobieczky , G.Dulikravich, and B.H.Dennis, *Parameterized geometry formulation for inverse design and optimization.* Proceedings of the 4th international conference on inverse problems in engineering. Brazil: Rio de Janeiro, 2002.

57. M.Klein and H.Sobieczky, *Sensitivity of aerodynamic optimization to parameterized target functions.* Proceedings of the international symposium on inverse problems in engineering mechanics. Japan: Nagano, 2001.

58. S.Sarakinos, I.Valakos, and I.Nikolos, *A software tool for generic parameterized aircraft design.* Advanced Eng Software, 2007. **38**(1): p. 39-49.

59. D.Rodriguez and P.Sturdza, *A rapid geometry engine for preliminary aircraft design.* Proceedings of the 44th AIAA aerospace sciences meeting and exhibit. Nevada: Reno; . 2006.

60. H.Ugail, M. I.G.Bloor, and M.J.Wilson., *Manipulation of PDE surfaces using an interactively defined parameterization.* Computers and Graphics. , 1999. **23**(4): p. 525-534.

61. E.Elyan and H.Ugail, *Interactive Surface Design and Manipulation using PDE-Method through Autodesk Maya Plug-in.* International Conference on CyberWorlds, 2009

62. Autodesk Maya. Understanding the Maya API - The Dependency Graph.

63. D.Gould, *Complete Maya Programming: An Extensive Guide to MEL and C++ API*.

64. M.J.Smith., *Maya Plug-In Power.* . 2008: Charles River Media. . 254.

65. J.Chang and J.J.Zhang, *Mesh-free deformations.* Computer animation and virtual words, 2004. **15**: p. 211–218.

66. H.Ugail and S.Kirmani., *Shape Reconstruction using Partial Differential Equations.* WSEAS Transactions on Computers., 2006. **10**(5): p. 2156-2161.

67. D.D.Hand, *Snow White and the Seven Dwarfs.* Walt Disney Animated Classics, 1937.

68. D.Sturman, *Interactive Keyframe Animation of 3-D Articulated Models.*, in *Graphics Interface '86, Tutorial on Computer Animation.* 1986.

69. D.Tost and X.Pueyo, *Human body animation: a survey*. Vol. 3. 1988: Springer Berlin / Heidelberg.

70. A.Watt and M.Watt, *Advanced Animation and Rendering Techniques - Theory and practice*.

71. M.Sipser, *Introduction to the Theory of Computation.* 1997: Course Technology Inc.

72. C.W.Reynolds, *Flocks, Herds, and Schools: A Distributed Behavioral Model.* Computer Graphics, SIGGRAPH '87 Conference Proceedings, 1987. **21**(4): p. 25-34.

73. A.Treuille, S.Cooper, and Z.Popovi´, *Continuum Crowds.* Proceedings of ACM SIGGRAPH, 2006. **25**(3): p. 1160 - 1168.

74. R.White, K.Crane, and D.A.Forsyth, *Data Driven Cloth Animation.* ACM SIGGRAPH SESSION, 2007.

75. D.Hinsinger, F.Neyret, and M.P.Cani, *Interactive Animation of Ocean Waves.* ACM-SIGGRAPH, 2002.

76. D.Terzopoulos, X.Tu, and R.Grzeszczuk, *Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World. Journal of Artificial Life, .* 1994.

77. R.Fedkiw, T.Aslam, and H.Jensen, *Visual Simulation of Smoke.* Siggraph Annual Conference, 2001: p. 23-30.

78. L. D.Landau and E.M.Lifshitz, *Fluid Mechanics*. 2 ed. 1998: Oxford: Butterworth-Heinemann.

79. A.Bruderlin and T.Calvert., *Goal-directed, dynamic animation of human walking. .* Proceedings of the 16th annual conference on Computer graphics and interactive techniques, 1989: p. 233 - 242

80. H.Ugail, *Spine based shape parameterization for PDE surfaces.* Computing, April, 2004. **72**(1-2).

81. C.J.Kuo, R.S.Huang, and T.G.Lin., *Synthesizing lateral face from frontal facial image using anthropometric estimation.* Proceedings of International Conference on Image Processing, 1997. **1**: p. 133-136.

82. J.E.Chadwick , D.R.Haumann, and R.E.Parent., *Layered Construction for Deformable Animated Characters.* Proceedings of the 16th annual conference on Computer graphics and interactive techniques. , 1989: p. 243 - 252

83. T.W.Sederberg and S.R.Parry., *Free-Form Deformation of Solid Geometric Models.* ACM SIGGRAPH 2008, 1986. **20**(4): p. 151-160.

84. L.P.Nedel and D.Thalmann, *Modeling and deformation of the human body using an anatomically-based approach.* Proceedings of the Computer Animation, 1998 p. 34.

85. F.Multon, L.France, and M.P.Cani-Gascuel, *Computer Animation of Human Walking: a Survey.* JVCA, 1999. **10**: p. 39-54.

86. H.Alaskari, *Msc thesis in Cyclic animation user interface in Maya.* 2005, University of Bradford.

87. D.Ormoneit, M.J.Black , and T.Hastie, *Representing cyclic human motion using functional analysis.* Elsevier, 2005. **Volume 23**(Issue 14): p. 1264–1276.

88. D.Ormoneit, et al., *Learning and Tracking Cyclic Human Motion.* The MIT Press, 2001: p. 894-900.

89. L.Ieronutti and L.Chittaro, *Employing virtual humans for education and training in X3D/VRML worlds.* Elsevier 2007. **Volume 49**(Issue 1): p. 93-109

90. X.Yang, D.C.Petriu, and T.E.Whalen, *Hierarchical Animation Control of Avatars in 3-D Virtual Environments.* IEEE. , 2005. **54.**(3.): p. 1333 - 1341.

91. P.Kalra, N.Magnenat-Thalmann, and L.Moccozet, *Real-Time Animation of Realistic Virtual Humans.* IEEE Computer Graphics and Applications, 1998: p. 42-56.

92. S.Il.Park and J.K.Hodgins, *Capturing and Animating Skin Deformation in Human Motion.* Proceedings of ACM SIGGRAPH., 2006. **25** (3): p. 881 - 889.

93. M.Pratscher, P.Coleman, and J.Laszlo, *Outside-In Anatomy Based Character Rigging.* Proceedings of the ACM SIGGRAPH, Los Angeles, California 2005 p. 329 - 338.

94. D.James and C.Twigg, *Skinning mesh animations.* International Conference on Computer Graphics and Interactive Techniques archive ACM SIGGRAPH. , 2005 p. 399-407.

95. I.Baran and J.Popovi´c, *Automatic Rigging and Animation of 3D Characters.* ACM Transactions on Graphics. Proceedings of the 2007 SIGGRAPH conference, July 2007. **26**(3): p. 8.

96. C.Hecker, B.Raabe, and R.W.Enslow, *Real-time Motion Retargeting to Highly Varied User-Created Morphologies* ACM SIGGRAPH 2008 2008: p. 1-11.

97. Q.Zhu, et al., *Three-dimensional flow structures and vorticity control in fish-like swimming.* Journal of Fluid Mechanics. , 2002. **468.** : p. 1-28.

98. M.J.Lighthill, *Hydromechanics of aquatic animal propulsion. Annual Review of Fluid Mechanics* 1969: p. 413-446.

99. H.C.Lee, et al., *Development of Real Time Virtual Aquarium System.* IJCSNS International Journal of Computer Science and Network Security, July 2006. **6**(7A).

100. G.Allard, *Control of a Free-swimming Fish Using Fuzzy Logic.* The International Journal of Virtual Reality, 2007. **6**(3): p. 23-28.

101. M.Gleicher, *Retargetting Motion to New Characters.* Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998: p. 33 - 42

102. J.Monzani, P.Baerlocher, and R.Boulic, *Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting.* EUROGRAPHICS 2000, 2000. **19**(3): p. 11 - 19.

103. S.R.Marschner, B.Guenter, and R.Raghupathy, *Modeling and Rendering for Realistic Facial Animation.* Proceedings of the Eurographics Workshop on Rendering Techniques. , 2000: p. 231 - 242

104. F.I.Parke, *Computer fenerated animation of faces.* SIGGRAPH ACM 1972. **1**: p. 451- 457.

105. Z.Deng and J.Noh, *Computer Facial Animation: A Survey* in *Data- driven 3D facial animation*. 2007, Springer.

106. P.Joshi, et al., *Learning Controls for Blend Shape Based Realistic Facial Animation.* Eurographics/Siggraph ACM., 2006(17).

107. J.P.Lewis, et al., *A User Interface Technique for Controlling Blendshape Interference*, in *Data-driven 3D facial animation*. 2007, Springer.

108. K.Arai, T.Kurihara, and K.Anjyo, *Bilinear interpolation for facial expression and metamorphosis in real-time animation.* The Visual Computer. , 1996. **12**(3): p. 105-116.

109. K.Waters and M.T.Levergood, *An Automatic Lip-Synchronization Algorithm for Synthetic Faces.* Proceedings of the second ACM, 1994: p. 149 - 156.

110. F.I.Parke and K.Waters, *Facial Action Coding System.* Consulting Psychologist Press., 1977.

111. I.S.Pandzic and R.Forchheimer, *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. 2002: John Wiley and Sons.

112. S.Coquillart, *Extended free-form deformation: a sculpturing tool for 3D geometric modelling.* Proceedings of the 17th annual conference on Computer graphics and interactive techniques, 1990: p. 187 - 196.

113. H.Du and H.Qin, *Free-Form Geometric Modeling by Integrating Parametric and Implicit PDEs. .* IEEE Transactions on Visualization and Computer Graphics, 2007. **13**(3).

114. S.Platt and N.Badler, *Animating facial expression.* ACM SIGGRAPH., 1981. **15**(3): p. 245 - 252

115. S.Platt, *A structural model of the human face.*, in *Department of Computer & Information Science.* 1991, University of Pennsylvania.

116. K.Waters, *A muscle model for animating three-dimensional facial expression. .* International Conference on Computer Graphics and Interactive Techniques, 1987: p. 17 - 24

117. J.Chen, *Automatic face animation with linear model.*

118. D.Terzopoulos and K.Waters, *Physically-Based Facial Modeling, Analysis,and Animation. .* International Conference on Computer Graphics and Interactive Techniques., 1995: p. 55-62.

119. K.Kähler, J.Haber, and H.P.Seidel, *Geometry-based Muscle Modeling for Facial Animation . .* Graphics interface 2001 2001: p. 37 - 46

120. R.Lee, D.Terzopoulos, and K.Waters, *Realistic Modeling for Facial Animation.* Proceedings of the 22nd annual conference on Computer graphics and interactive technique., 1995: p. 55 - 62.

121. A.Fedorov, et al., *Talking Head: Synthetic Video Facial Animation in MPEG-4.* International Conference Graphicon Moscow, Russia, 2003.

122. K.Balcı, *Xface: MPEG4 Based Open Source Toolkit for 3D Facial.* Proceedings of the working conference on Advanced visual interfaces, 2004: p. 399 - 402

123. S.W.Kim, et al., *A Talking Head System for Korean Text.* World Academy of Science, Engineering and Technology, 2005. **50**.

124. R.Pasquariello and C.Pelachaud, *Greta: A Simple Facial Animation Engine.* In Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications., 2001.

125. Y.Huangm, et al., *Real-time Lip Synchronization Based on Hidden Markov Models.* The 5th Asian Conference on Computer Vision, Melbourne, Australia., January 2002.

126. S.Maddock, J.Edge , and M.I.Sanchez, *Movement Realism in Computer Facial Animation.* Workshop on Human-animated Characters Interaction., 2005: p. 4.

127. S.R.Wallace, *The Anatomy of A.L.I.C.E.*

128. T.Dutoit, *High-quality text-to-speech synthesis* 2001: Springer.

129. M.Turing, A., *Computing machinery and intelligence.* Mind., 1950. **59**(236.): p. 433-460.

130. A.M.Galvão, et al., *Adding Personality to Chatterbots Using the Persona-AIML Architecture*, in *Book Advances in Artificial Intelligence*. 2004, Springer Berlin / Heidelberg. p. 963-973.

131. R.Wallace. *Artificial Intelligence Markup Language (AIML) v1.0.1*. AI Foundation Working Draft 2001.

132. N.Mana and F.Pianesi, *HMM-based Synthesis of Emotional Facial Expressions during Speech in Synthetic Talking Heads.* Proceedings of the 8th international conference on Multimodal interface., 2006: p. 380 - 387.

133. A.M.Galvão, et al., *Persona-AIML: An Architecture for Developing Chatterbots with Personality.* Third International Joint Conference on Autonomous Agents and Multiagent Systems - New York, USA, 2004. **3**.

134. T.D.Giacomo, S.Garchery, and N.M.Thalmann., *Expressive Visual Speech Generation.*, in *Data-driven 3D facial animation.* 2007, Springer.

135. A.M.Fonte, *TQ-Bot: An AIML-based Tutor and Evaluator Bot Fernando* Journal of Universal Computer Science, Austria. **15**(7).

136. J.Haber , et al., *Face to Face: From Real Humans to Realistic Facial Animation.* Proceedings Israel-Korea Binational Conference on Geometrical Modeling and Computer Graphics 2001, Oct 2001: p. 73-82.

137. Y.Sheng, et al., *PDE-Based Facial Animation: Making the Complex Simple* Proceedings of the 4th International Symposium on Advances in Visual Computing, 2008: p. 723 - 732.

138. F.Pighin, et al., *Synthesizing Realistic Facial Expressions from Photographs.* Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998: p. 75 - 84