Real Time Morphing of Polyhedra

Author: James Gilbert

Supervisor: Dr. Steve Maddock

Module Code: COM3010

Date: 5th May 2010

This report is submitted in partial fulfilment of the requirement for the degree of Master of Engineering in Software Engineering by James Robert Gilbert.
Signed Declaration

All sentences of passages quoted in this report from other people’s work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: James Gilbert

Signature ...........................................

Date ...............................................
Abstract
The effect of morphing from one object to another has been used as special effect heavily in major Hollywood films for the past two decades. However these approaches have traditionally required animators to orchestrate the interpolation of object aspects to create a morph. There have been many morphing approaches developed to produce real time results, however most still require user input and produce ridged interpolation. This report proposes a real time solution to morphing by fusing boundary and volumetric morphing to produce interpolation between objects, with the focus on creating accurate morphing of two objects more akin to the animations produced by the movie entertainment industry but without user intervention with the advantage of being fully automatic.
Acknowledgments

I would like to thank my supervisor Steve Maddock for his guidance and support which helped me produce a project I am proud of. I would also like to thank:

- My girlfriend Sarah for keeping me in hysterics throughout the entire project.

- My housemates, for constantly keeping me on my toes to produce a project of high programming standard.

- Paul Richmond for his help getting started with CUDA GPU programming.

Contents

Signed Declaration ...................................................................................................................... i
Abstract ................................................................................................................................... ii
Acknowledgments .................................................................................................................. iii

Chapter 1: Introduction ............................................................................................................... 1
1.1: Game application ............................................................................................................... 1
1.2: Overview of subsequent chapters ................................................................................... 1

Chapter 2: Literature Review ..................................................................................................... 3
2.1: 3D Object Representation ............................................................................................... 3
2.1.1: Boundary Object Representation ............................................................................... 3
2.1.2: Volumetric Object Representation ........................................................................... 4
2.1.3: Sphere Approximations ............................................................................................. 5
2.2: Correspondence .................................................................................................................. 6
2.2.1: Boundary Correspondence ........................................................................................ 6
2.2.2: Volumetric Correspondence ...................................................................................... 9
2.3: Interpolation ....................................................................................................................... 9
2.3.1: Boundary Interpolation ............................................................................................ 9
2.3.2: Volumetric Interpolation .......................................................................................... 10

Chapter 3: Requirements and Analysis ........................................................................................ 12
3.1: Stage 1 – Sphere Approximation of boundary object ..................................................... 13
3.1.1: Octal approach .......................................................................................................... 13
3.2: Stage 2 – Metaball Approximation .................................................................................. 13
3.2.1: Step 1 – Dividing the object space .......................................................................... 13
3.2.2: Step 2 – Placing of metaball centres ....................................................................... 13
3.2.3: Step 3 – Calculating metaball influence fields ......................................................... 13
3.2.4: Step 4 – Skinning the object .................................................................................... 14
3.3: Stage 3 – Boundary Morph from Source to Metaball approximation .......................... 15
3.4: Stage 4 – Volumetric morph between Source and target Metaball approximations .... 15
3.4.1: Volumetric Morph Techniques ................................................................................ 15
3.4.1.1: Intermediate Sphere ........................................................................................... 15
3.4.1.2: Explosion Morph ............................................................................................... 15
3.4.1.3: No Source Object .............................................................................................. 16
3.4.1.4: Puddle Effect .................................................................................................... 16
6.3: Morphing In-Betweens Generation Results (Requirements reference 2.1&3) .......... 41
6.3.1: Morph Test 1 (Teapot -> Penguin)................................................................. 41
6.3.2: Morph Test 2 (Penguin -> Suzanne)............................................................... 42
6.3.3: Morph Test 3 (Suzanne -> Torus)................................................................. 42
6.3.4: Morph Test 4 (Torus -> Teapot)..................................................................... 43
6.4: Intermediate Mesh Morphs (Requirements reference 3.1-3.4)........................ 44
6.5: Original Specification Comparison..................................................................... 45
6.6: Testing on more complex Game Models............................................................. 45
6.7: Performance Results.......................................................................................... 47
6.8: Morphing Showcase Game (Requirement reference 4)....................................... 48
Chapter 7: Conclusion .............................................................................................. 50
7.1: Overview............................................................................................................. 50
7.2: Implementation................................................................................................... 50
7.3: Future work....................................................................................................... 50
References .............................................................................................................. 51

Glossary

Object - An object is any shape which can be created either 2d or 3d, for example a square or a cube.
Vertex – A point in two or three-dimensional space.
Edge – A line which joins two vertices.
Face – A plane connected by vertices and edges.
Source Object - an object from which a metamorphosis animation will start.
Target Object - desired final shape after a metamorphosis has been run.
Topology - The topology of an object refers to how the vertices, edges and faces are connected to describe its shape.
Homeomorphism - If the topology of one object matches that of another they are said to be homeomorphic. That is, if there is a one-to-one correspondence between the points on the surfaces of the two objects exits.
In-betweens – Generated animation frames between the base and target positions during interpolation.
Vertex interpolation - corresponding vertices are transformed, from the base position to the corresponding end position in a variable number of steps.
Voxel - Voxels are to 3D graphics what pixels are to 2D. Whereas pixels are 2 dimensional squares of colour data which constitute an image, voxels are 3D cubes which placed together cover the entire volume of an object.
Chapter 1: Introduction

The morphing of two graphical objects is the process of gradual transition from source to a target state. As long as the overall structure of the morphing objects are preserved to some extent during the transitional process the object animation can be as complex or ridged as desired. Morphing algorithms were introduced in early 1990; these algorithms were 2D and offered generation of image in-betweens. The 2D implementation transformed the position of the pixels while also altering their colour property. The technique was used to create the morphing effects in the 1991 music video Black or White (Landis, 1991). Morphing of three dimensional objects is extremely popular in the movie industry, and has appeared in such titles as Xmen (Singer, 2000), Harry Potter and the Philosopher’s stone (Columbus, 2001) and The Matrix Reloaded (Wachowski, 2003) to name a few. This paper gives an overview of many of the existing methods for producing three dimensional morphs then continues to propose a technique to attain three dimensional morphing between polyhedra which can be executed in real time. The application of such a technique lends itself to gaming applications where speed is of paramount importance to give a smooth gaming experience. There are three major areas to address in order to complete a successful morph outlined in Table 1.1. These areas will be described in more detail and discussed for direction of a new system.

<table>
<thead>
<tr>
<th>Representation</th>
<th>There are many techniques for representing a three dimensional object, each possessing different methods for display.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correspondence</td>
<td>Once represented attributes of the source object must be matched to the target object. This matching is required in order to gain an interpolation. However, deciding how two polyhedra likely to be radically different correspond to each other isn’t a trivial task.</td>
</tr>
<tr>
<td>Interpolation</td>
<td>Interpolation is the generation of in-betweens; these are generated animation frames between the source and the target. This is the transformation of the corresponding features from the source position to the target.</td>
</tr>
</tbody>
</table>

Table 1.1 Outlines the vital components needed to create a morph from one object to another.

1.1: Game application

As part of the report a game showcasing the proposed morphing technology is required. To keep the game simple an existing well established children’s card game will be adapted to incorporate morphing objects into its rules. The game will be used to showcase the newly developed morphing system in a real application.

1.2: Overview of subsequent chapters

**Chapter 2. Literature Review**: Review and discussion of previous work related to the field of morphing.

**Chapter 3. Requirements and analysis**: New morphing system is presented in detail along with a detailed evaluation method for evaluation of the success of the system implementation.

**Chapter 4. Design**: Outlines design choices of the morphing system, and presents UML diagrams explaining the functionality and structure of the code.
Chapter 5. Implementation and testing: Examines in detail the major algorithms used within the implemented morphing system. Test results on aspects of the system are evaluated, and are used to justify additional design choices.

Chapter 6. Results and discussion: The final outputs from the morphing system are presented and are critically evaluated using the principles outlined in the evaluation section. The overall success of the methods used to create a morph is discussed and Improvements to the system are discussed.

Chapter 7. Conclusion: Final thoughts on the success of the project, and discussion on the direction of possible future work.
Chapter 2: Literature Review

There is a wealth of literature on morphing therefore constructing a whole review of this field would be an overwhelming amount of information, many of which being of small impact. This literature review instead highlights only the papers thought to have a greater importance to the project. Starting with a critical analysis of the processes needed for a morph, the reviewed papers are divided into three sections:

- **Representation** – How three dimensional objects can be represented to create shapes?
- **Correspondence** – How aspects of these objects can be corresponded to one another?
- **Interpolation** – How these aspects can move from one point to another to create a morph?

Each section will discuss previous work and evaluate its importance in the creation of a new morphing technique. There are two main techniques which are used to create shape changing animations, boundary morphing and volumetric morphing.

Boundary morphing techniques refers to objects defined by vertices on their boundary; these vertices are then interpolated to new positions to represent the target object. Boundary morphing consists of two stages:

1.) Generation of vertex correspondence: Attaining a one to one correspondence between source and target objects.
2.) Vertex interpolation: The generation of in-betweens.

Volumetric morphing approaches are used for the metamorphosis of objects defined by a density function. Therefore in order to transition between objects the density is interpolated. The issue of correspondence is removed, simplifying the procedure. However, standard models are represented by their boundary so volumetric alternative would need to be generated. The subsequent sections discuss these techniques in greater detail to give a better understanding of their application.

2.1: 3D Object Representation

Polyhedra in 3D graphics although similar to real world objects by their appearance, can’t be thought of in the same way. Whereas real objects are made up of solid material, computer generated representations are typically made up of structural data which describes boundary of the final shape. To successfully display an object, this representation must be determined.

2.1.1: Boundary Object Representation

For boundary representation three dimensional objects are created by vertices placed in the world space interconnected to form polygons. The interconnected polygons can be placed in such a way to produce recognisable polyhedral objects. This technique is the most common way of representing three dimensional shapes in computer graphics. Computer games use
this approach to create entire navigable worlds. Polyhedral objects usually consist of the following attributes

- A list of vertices, containing Cartesian X Y and Z coordinates.
- A list of faces referencing how the above vertices are connected to form flat surface.
- Normals for each vertex to setup correct lighting for the object.
- Texture mapping UV coordinates to assign colour to the model.

Normals can be defined per face or per-vertex. A face normal is a vector perpendicular to the surface of the face. A vertex normal can be calculated by taking a mean of all the surrounding face normals; Figure 2.1 shows an icosphere with face (blue) and vertex (red) normals. The effect on the shading of an object using each technique is clear. The vertex normal approach gives a smooth looking surface, which gives illusion of a rounder object, whereas face normals produce angular shading making the structure of the object more identifiable.

2.1.2: Volumetric Object Representation

A volumetric object is described by density function rather than boundary information. In order to gain a solid object the volumetric data must be generated in a structured manner, and then manipulated into a boundary mesh usually specified by some threshold cut-off value of the density information. The steps required to create volumetric objects are outlined in the following subsections, this is not a complete survey of such objects, and instead concentrates on the methods thought useful to a morphing application.

2.1.2.1: Matrix Representation

Volumetric objects can be created by a number of neighbouring cubes tightly placed together called Voxels, much like building blocks. To achieve this the world space has to be divided into a three dimensional grid where each voxel is given a value of either on or off calculated by the density function, on referring to voxels inside the density field. These ‘on’ cubes are then rendered, giving an approximation of an object. The quality of the object produced is based on the resolution of the voxel space grid. A higher resolution gives smoother results but suffers from increased computational overheads.

2.1.2.2: Implicit Surfaces - Metaballs

Metaballs or Blobby Objects produce an object with a gooey organic quality and were first proposed by Blinn in the early 1980s (Blinn, 1982). The effect has been used in films like Terminator 2 (Cameron, 1991) to produce transitions from a human form into other objects in a fluid motion. The ease at which this kind of structure can change shape to form alternative layouts makes it appealing for use in morphing. Metaballs are calculated by the following process:
1. Divide the object space into a three-dimensional voxel matrix.
2. Place particles in the object space with associated magnitudes
3. Calculate an influence at each vertex in the voxel matrix.
4. Use the vertex influences to skin the newly formed object

2.1.3: Sphere Approximations
As computer technology advances the need for more complex models has been required. As the resolution of vertices is increased the processor load required to perform per vertex calculations starts to grow to an unmanageable size for real time computer graphics. By using adequate simplified approximations of objects, computation required can be reduced significantly. This approach is used for many areas of computer graphics to speed up computation, such as collision detection (Hubbard, 1996) and ray tracing. There is also an opportunity to use sphere approximations to create volumetric representations from polyhedra represented by boundary values, reducing the number of points to be interpolated for a morph. Volumetric morphing is only possible with an appropriate way to attain adequate volumetric representations of polyhedra; sphere trees give the required stepping stone in production of such a representation.

2.1.3.1: Octal Approach
This recursive approach (Liu et al, 1988) to sphere approximation only requires an inclusion algorithm. First a boundary object is placed inside a bounding sphere, which is then recursively dividing the space inside sphere into smaller spheres. The new spheres are tested for inclusion inside the original polyhedra, those external to which are disregarded, the remaining internal are processed using the same dividing algorithm. This procedure is repeated until a specified desirable level of detail has been achieved. This approach offers a simple method for the generation of sphere approximation. However the positioning of the spheres is ridged and so doesn’t always produce the most efficient structures.

2.1.3.2: Hubbard Approach
Needing a much tighter representation of the original object than that created by the octal approach, (Hubbard, 1996) introduced a new method for approximating objects with spheres. The approach addresses the issue of accuracy of sphere placement. The paper claims the results closely approximate where a human may place the spheres in order to create an object, giving a more accurate representation. The technique for creation of the hierarchies is much more complex than for octree generation and will be briefly explained in this section.

Creation procedure of a Hubbard sphere tree

- Figure 2.2 (b): P number of points are placed on the polygons surface evenly, the value P is supplied by the user
- Figure 2.2 (c): A Voronoi diagram is then calculated (Hubbard, 1995) based on the P points. The Voronoi vertices which lie inside the polyhedron make up a medial-surface, which is achieved by applying an inclusion testing algorithm.
**Chapter 2: Literature Review**

- **Figure 2.2 (d):** Each internal Voronoi vertex represents a sphere center. The points which affect this position (3 for 2D example) are used to specify the sphere radii values, these points are referred to as the forming points.

![Figure 2.2 a) Source polygon, b) Points placed equidistance around perimeter, c) Voronoi diagram attaining medial axis (black), d) Circles generated using Voronoi vertices and forming points.](image)

The spheres created are the highest resolution approximation for the shape. Combining these spheres produces a structure of lesser resolution and is used as the parent to the combined spheres. This is the opposite approach to tree creation than taken by the octal approach. Therefore, if a low resolution was desired the Hubbard approach would calculate the most complicated tight fitting structure first and work to simpler implementations until the desired resolution was reached. The octal tree on the other hand would start simple and subdivide to the desired resolution. So would produce a sphere structure in less time.

The tree structure was desirable for Hubbard who was using the approximation to simplify collision detection to cut down the number of ray collision test required. Applied to morphing this structure isn't as useful. Aside from linking the resolution with draw distance, any morph will just be a morph between two trees at the same depth in the tree. Therefore it will just be a morph between two lists of spheres rather than take advantage of the tree hierarchy. The generation of a medial-axis surface gives a structure to the spheres, and stores the topology of the overall object. Using this approach would therefore have access to this topological information which could be beneficial in preservation of attributes during a morph.

**2.2: Correspondence**

In morphing the idea of correspondence is central to how in-betweens are generated. As a morph is a gradual move from one state to another, the components which make up the structure of the start and end states must first be recognised as corresponding to each other. The subsequent subsections examine this problem in more detail, and offer description and discussion on existing methods used.

**2.2.1: Boundary Correspondence**

To morph from one shape defined by boundary to another, homeomorphic correspondence of vertices is required. To achieve homeomorphism there is required to be a one-to-one correspondence between the vertices on the surface of the source and target objects. Without this correspondence an interpolation between the two will produce holes. Unless specially designed, finding two objects which are made from the exact number of vertices is very rare. This report aims to successfully morph objects consisting of any number of vertices.
with any other arbitrary object. Therefore the vertex correspondence problem must be addressed as part of the morphing process. In order to gain homeomorphism between the source and target polyhedra, the structure of one or both must be altered. To gain homeomorphism the following techniques may be applied:

- Addition of vertices (subdividing) – Introduces new vertices and rearranges the structure to correctly accommodate them.
- Removal of vertices (decimating) – Removes vertices from the polyhedra and collapses any structures affected to ensure no holes in the object suffice.
- Intermediate object – An object which aims to correspond to both source and target object and act as an intermediate transition.

2.2.1.1: Intermediate Objects
Intermediate objects add an additional stage to object morphing, instead of going straight from the source to target; a third object is introduced mid interpolation. Morphing to an intermediate object helps solve the problem of morphing non-homeomorphic objects. All that need be created is a transition from the source object to a sphere, together with a transition from the target object to a sphere played in reverse. The source generated sphere is simply just replaced by the generated target sphere. However if one of the meshes has a low vertex count the profile of the sphere can look very different to one with a much higher resolution. This sudden jump in resolution could be an obvious unsmooth change in shape, and so wouldn’t result in a smooth morph.

Projecting the vertices of polyhedron into a uniform sphere successfully isn’t a trivial task. A simple approach could have the vertices following the direction of a vector originates at the center of a polyhedron pointing in the direction of the vertex’s source position and moving out to a certain radius. This approach however will only be successful if the polyhedron is a so called Star polyhedron. That is, there exists a point inside the object which has line of sight of all the vertices. This point known as the star point would have to be used as the objects centre. Non-star polyhedra require a more sophisticated procedure for unwrapping calculation, such as the star skeleton approach of (Shapira and Rappoport, 1995). Figure 2.3 illustrates the edge collision danger when projecting vertices outwards to create a circle in 2D space.
2.2.1.2 Vertex relaxation
As can be seen in Figure 2.3 even if the vertices are correctly projected onto a unit circle they don’t necessarily produce a circular shape. In order to achieve this, the points must be equidistance from one another, referred to as vertex relaxing. (Turk, 1991) proposed a vertex relaxation technique which he used for the generation of textures for arbitrary surfaces. Crudely explained, a user defined number of points are placed randomly onto the original surface. Each point then pushes other points around the surface by repulsion of one another. This repulsive force falls off linearly. The repulsive radius is calculated by Equation 2.1, if this value is less than the distance between two points, the two points don’t affect one another.

\[ r = 2 \sqrt{\frac{a}{n}} \]

Equation 2.1 Repulsive radius equation

Where: \( a \) = area of surface, \( n \) = number of points on surface.

2.2.1.3 Creating new Source and Target objects using Intermediate Objects (Kent, 1992) used the intermediate sphere approach to create new versions of the source and target objects in order to gain homeomorphic objects. The vertices of each model were projected outwards 'like a balloon' to create a unit sphere for each. These spheres were then used to create new source and target objects from the two combined unit spheres, achieved by clipping the edges of the source unit sphere by the target. The generated source and target meshes now exhibit homeomorphism and have vertex correspondence; Figure 2.4 shows this procedure in 2D space. The morph is achieved by a linear interpolation as in Figure 2.5. This approach is effective when dealing with just two objects; however it becomes increasingly memory intensive when adding additional objects. For each object there must exist a version which matches the topology of every other object to be morphed with. The number of required generated objects can be calculated by applying Equation 2.2.

**Number of generated objects** = \( n^2 - n \)

Equation 2.2 Number of required generated objects per object \( n \) in scene.

![Figure 2.4 Generation of new source and target objects which possess homeomorphism.](image)
Alternatively the intermediate object can be programatically generated using the data from both the source and target meshes. (Lee et al, 1992) were able to create a smooth transition from source to target through a generated intermediate mesh without it being noticeable during the interpolation. This approach successfully reduces the number of extra generated objects in the scene per object; however the problem is still present just to a lesser extent.

2.2.1.4 Star-Skeleton Representation

It is unlikely that the source and target objects are going to be star polyhedra. To address the problem of dealing with non-Star polygons the Star-Skeleton Representation was developed by (Shapira and Rappoport, 1995). Skeletons of connected Star-Points corresponding to a collection of Star-Pieces’ and midpoints between connecting ‘Star-Pieces’ are first created for each polygon. The two Star-Skeletons are then ‘unfolded’ which refers to linearly interpolating the points from the source to the target position. The Star-Skeleton interpolation uses edge-angle interpolation (Sederberg et al, 1993). Figure 2.6 shows an example of a Star-Skeleton representation for an arbitrary polygon.

2.2.2: Volumetric Correspondence

Metaballs gain their volumetric information from the individual points and magnitudes associated which combine to produce the area of influence. Correspondence of two such structures would be the matching of these points. Thinking of correspondence as the matching of disconnected points is very desirable for creating a highly responsive morphing algorithm. Metaball centres may be simply matched to their closest corresponding centre in the target. Homeomorphism is then simply a case of removing unwanted centres or introducing new ones slowly to create smooth transitions.

2.3: Interpolation

Effective interpolation is an important aspect of the morphing algorithm. This component is directly visible to an observer in the end product and so has a high impact on the overall aesthetics of a morph.

2.3.1: Boundary Interpolation

Polyhedra described by their boundary have three main components, vertices, edges and faces. These interconnected components produce many issues when attempting to transform their positions. This section introduces some of the major issues in interpolation of these structures, and presents previous work in the field of boundary interpolation.
2.3.1.1: Vertex Interpolation
The simplest approach to vertex interpolation is a linear transition from the source object vertex Cartesian coordinate to the corresponded target vertex position as a function of time. An alternative to linear interpolation is to map the trajectory to a generated curve such as a Bezier curve. The Cartesian position of the vertex is acquired from applying a function and varying time t. More complex animation curves could use the source vertex coordinate as the start of the curve, an intermediate point as the mid frame of the curve and the target coordinate position as the end position. This interpolation along a curve will give more interesting varied results for the generated in-betweens. Though computationally light the transition is likely to give unsatisfactory interpolation results which contain distortion of shape, and edge collisions, seen in Figure 2.3. These issues arise as the vertex path doesn’t have any knowledge of its connecting structure of vertices.

2.3.1.2: Edge Angle interpolation
(Sederberg et al, 1993) took a revised approach to vertex interpolation in order to address the problem of preserving the shape of the source and target objects. Instead of ignoring the structural information and interpolating an object as a set of Cartesian vertex information, the paper proposed that an object could be treated as a set of edge lengths and the angles between them. By altering the angles and length of edges the volume of the shape is preserved to a much higher degree; Figure 2.7 shows an example from the paper of an interpolation path of an arm, contrasting linear and edge-angle interpolation approaches. The linear path produces edge intersection and loses the original shape of the arm. The edge angle approach produces results which are recognisable as arm movements keeping the volume of the original source more constant and preserving length. For situations where the source and target objects or parts of them are affine transformations the edge angle interpolation technique gives successful results, where successful here means increased feature preservation. This approach requires similar homeomorphic objects and focuses on using morphing as a means to create in-betweens for similar objects to create animations between them, such as making a figure appear to run by just supplying key frames and interpolating between them.

2.3.2: Volumetric Interpolation
Interpolation between metaball centres of influence is an easier task than boundary. Unlike boundary information, the centres have no interconnected structure therefore changes in one don’t have any influence on the remaining points. This results in the issues of edge intersection being removed as the points are not directly visible by the observer, only their influence is.
2.3.2.1: Fourier Volume Morphing
(Hughes, 1995) Examined the source and target objects in the Fourier domain; producing an algorithm he called Scheduled Fourier Volume Morphing. Disregarding the general shape of the original objects and instead interpolating the Fourier transform data, he was able to produce relatively smooth transition between volumetric objects. The ‘Schedule’ part of the algorithm refers to the order in which the frequencies present in the objects are interpolated:

- Gradually remove the high frequencies from the source object,
- Interpolate over the low frequencies of the target object,
- Blend in the high frequencies of the target object.

However fully automated, using this method it is difficult for the programmer to manipulate the end resulting morph. There isn’t enough flexibility in the design of the produced morph due to the generation being Fourier based.
Chapter 3: Requirements and Analysis

The proposed morphing system is a combination of boundary and volumetric morphing techniques. The system overview is presented in Figure 3.1. The morphing system has been divided into 5 separate stages which fall into two categories, preprocess stages (1,2) and run morph path stages (3,4,5). This section will describe in more detail how these 5 stages will be realised and go on to explain the way in which they will fit together to create a complete morphing solution.

Figure 3.1 Full outline of proposed morphing system, showing both initialisation and runtime events.
3.1: Stage 1 – Sphere Approximation of boundary object
The Source and Target objects require an intermediate representation to be generated at initiation of the program. This representation is a sphere approximation as considered in Section 2.1.3. The procedure for creating this representation will be an adaption of the octree approach due to the ease of implementation of the algorithm desirable for the creation of a large interconnected system such as this, however time permitting it would also be of interest to test the sphere tree approximation algorithm of (Hubbard, 1996) to produce tighter sphere approximations.

3.1.1: Octal approach
The process of sphere approximation is a recursive algorithm which, given a resolution recursively subdivides a bounding sphere to smaller subsequent ones. This stage of the algorithm is a pre-process so efficiency isn’t an issue. The main focus is instead on gaining an accurate approximation. The sphere approximation must resemble the original polyhedra to give an accurate morphing procedure in the later stages. Differing octree generation resolutions will be experimented with to produce the best possible sphere approximation which works well with the remaining stages of the morphing system.

3.2: Stage 2 – Metaball Approximation
Stage 2 requires the generation of an implicit surface which approximates the original source model. This implicit structure will be achieved using metaballs. For ease of explanation this stage will be further deconstructed into 4 separate steps.

3.2.1: Step 1 – Dividing the object space
The object space is divided into a voxel matrix. The values for the matrix are specified by the user, giving a variable level of resolution. The higher the resolution of this matrix the more the program will suffer from computational load, however the smoother and more aesthetically pleasing the end representation will be.

3.2.2: Step 2 – Placing of metaball centres
The metaball data will come directly from sphere tree data. Each sphere will describe a metaball, the radii of the spheres will be used as a function of the magnitude (or influence) of the metaballs. This function will be altered during the testing stage of the project to achieve the most desirable effect.

3.2.3: Step 3 – Calculating metaball influence fields
Metaball influence is calculated for each vertex in the voxel matrix. This influence is the combined influence of all the particles in the scene. How this influence is calculated for the metaballs can give different behavioural effects. The formulae used by (Blinn, 1982) in his original proposal for creating metaball objects are described below. The value stored at each vertex is converted to a Boolean, indicating if it is within the area of influence of the overall object density.
Chapter 3: Requirements and Analysis

\[ D(x, y, z) = \sum_i b_i \exp(-a_i r_i^2) \]

Equation 3.1 Points which lie inside surface

\[ a_i = -\frac{\ln(T/b_i)}{R_i^2} \]

Equation 3.2 Standard deviation calculation

\[ b_i = T \exp(-B_i) \]

Equation 3.3: Height calculation

Defines the points which lie inside the surface. \( T \) is a constant ‘threshold’ value acting as a cut-off point for the surface definition. This exponential gives a Gaussian bump centred at \( r_i \), \( b_i \) corresponding to the height and \( a_i \) is the standard deviation.

Where \( R_i \) is the radius of the particle.

\[ B_i \] is referred to as the “blobbiness” of the object. This parameter is responsible for how attractive the surrounding particles are to one another.

3.2.4: Step 4 – Skinning the object

The object at this point is a set of influenced vertices in the voxel matrix, this data must be converted into a 3D polyhedron. A simple way to do this is to use the matrix approach of Section 2.1.2.1. As already discussed, this technique will result in very blocky structures. Using the Marching Cubes algorithm (Lorensen et al, 1987) a more accurate surface can be generated. The Marching Cubes algorithm or Marching Squares for 2D space works by assessing each voxel in the matrix generating a triangle mesh which produces a surface taking into account the influenced vertices of the voxel. It then moves on (or marches) to calculate the next voxel, and so forth until the entire matrix has been assessed. To explain how the algorithm works it is easier to think in the 2D space. The possible surfaces which can be generated are a finite set. For example, a square is made up of 4 vertices so there are possible \( 2^4 = 16 \) combinations of possible vertex influences a square can have. However, due to rotational and mirror symmetry these 16 can be reduced to just 5 possible. Figure 3.2 shows these 5 tiles and through example illustrates how they can be used to construct an approximate surface for a curve drawn in 2D space.

![Figure 3.2 Approximation of a curve using the marching squares algorithm.](image-url)
3.3: Stage 3 – Boundary Morph from Source to Metaball approximation
The technique chosen will be a simple crossfading style approach. As the two objects are similar, transitioning from the boundary representation to the volumetric can be achieved by gradually resizing the metaballs, from 0 to the specified calculated influence values, while shrinking the target boundary object. The procedure is very simple and will execute smoothly, offering more computational power to other areas of the morphing system, such as metaball calculation. Though gains are made through simplicity, the technique may lack in aesthetic effect. Certain objects which contain a lot of flat surfaces may produce undesirable transitions due to the spherical nature of the metaballs. It is expected that this technique will be more successful for curved objects than objects which are made up of largely flat surfaces.

3.4: Stage 4 – Volumetric morph between Source and target Metaball approximations
The complex process of matching vertices of the source to the target is circumvented by using volumetric approximations of the objects. Instead the particles which represent the centres of the metaballs will be interpolated. Far fewer calculations are required in order to create the in-betweens using this metaball approach. Artefacts such as flipped normals and holes are also avoided with this technique as the meta ball skin generated by the Marching Cubes algorithm (Lorensen et al, 1987) will govern the polyhedral shape at all times.

3.4.1: Volumetric Morph Techniques
Interpolating the source volumetric components to the target position differently will give the morph process different effects. By adding additional variation to the method by which an object morphs, it is possible to incorporate additional interpolation effects. These effects could be appealing additions to the morph when used within a game. The following subsections briefly discuss possible additional effects which could be incorporated, and crudely outlines possible mechanics for their inclusion. Possible morph effects that will be implemented during the project:

- Intermediate sphere.
- Explosion morph.
- Morph using no source object, i.e. target is made up from no source information.
- Puddle like effect used for intermediate object.
- Vertex Interpolation Acceleration & Damping

3.4.1.1: Intermediate Sphere
Morphing using metaballs simplifies interpolation with an intermediate sphere. The process is simple due to the nature of metaballs being spherical in shape. The arrangement of metaballs decided by the sphere approximation algorithms need just be translated to a central position, their combined influence will act as a single sphere where extra balls can be added or removed by slowly shrinking or growing them into place. The metaballs can then be simply moved linearly to the target positions.

3.4.1.2: Explosion Morph
By using each metaball as a particle the positions could be governed by a particle engine which simulates explosion trajectories of the balls. This effect is then played in reverse with
appropriate balls added or removed to make up the target structure. This gives a very playful morphing effect, much like what one would expect to see in computer games.

3.4.1.3: No Source Object
The idea of morphing from nothing is a strange concept; however it has been used to in many Hollywood films such as the birth of The Sandman in Spiderman 3 (Raimi, 2007). By slowly introducing balls from various positions in space to their correct position in the world, it is possible to slowly build up an object.

3.4.1.4: Puddle Effect
An effect similar to one used in the Terminator series of films (Cameron, 1991) for morphing liquid metal into other objects. The effect could be created in much the same way as the intermediate sphere, manipulating the metaball centres to a flat puddle shape mid morph.

3.4.1.5: Vertex Interpolation Acceleration & Damping
By specifying physical attributes such as mass to the metaballs, they can be programmed to interpolate in a much smoother fashion by calculating accelerations rather than straight linear speed. Metaballs with mass and acceleration can also be programmed to overshoot or oscillate about their targets. Applying standard engineering damping equations of Equation 3.4 to attain under damped results, the metaballs would oscillate with amplitude gradually resting at the target position, like a spring motion. Graph 3.1 shows different damping results from applying different damping coefficients.

\[ x = Ae^{-yt} \cos(w_1 t + \theta) \quad w_1 = \sqrt{(w_0^2 - y^2)} \quad w_0 = \frac{k}{m} \quad y = \frac{b}{2m} \]

Where,  
\( x \) = Displacement, \( k \) = Spring Constant, \( m \) = Mass, \( A \) = Amplitude, \( \theta \) = Phase, \( w_0 \) = Natural frequency, \( w_1 \) = Damped frequency, \( y \) = Damping factor, \( b \) = Damping coefficient

Equation 3.4 Damping equations

![Graph 3.1 Harmonic Oscillation results applying differing damping coefficients.](image-url)
3.5: Stage 5 - Boundary Morph from Target metaball approximation to Target

The final stage in the morphing procedure is similar to stage 3, it can be thought of as the reverse of which. The volumetric representation is contracted by decreasing the influences of the metaballs which create it. As the volumetric representation continues to shrink, revealing the target boundary object.

3.6: Showcase Game

The showcase demo game which will be created is a simple game of Pairs, traditionally played with cards. In a traditional game of Pairs the player is presented with face down cards. The cards are made up of sets of pairs randomly placed in a grid. The player must identify these pairs by choosing an initial card and attempting to find its matching pair. If found these cards are removed from the grid, if not both cards are returned to their original face down position. The morphing showcase game replaces the cards with sphere shaped 3D objects. These objects are used as the source object for morphs, when selected an interpolation reveals the target object, similar to turning a card in the original game.

3.6.1: Additional Rules:

Some of the additional morphing effects will also be incorporated into the design of the game. These effects will be introduced by rules not present in the traditional game:

- The game will be timed to introduce a possibility of losing.
- The interpolation time of the morph will be dependent on how many times the player has requested to see the hidden true object. At the start of the game transition will be quick, however consistently revealing the target object slows down the process taking up valuable seconds.
- Revealing an object too many times will cause it to explode, so the player cannot get maximum points.
- If the player successfully identifies the pairs they appear to melt into a puddle.
- In order to precede to the next level a certain number of pairs must be matched before the time runs out. Additional time and objects are added upon level up

3.7: Requirements Table

The requirements for the morphing system have been split up into three categories:

**Critical Tasks:** Vital components which must be implemented in order to create a fully functioning base system

**Desired Tasks:** Components which would create more desirable morphing effects, but not critical to the completion of the project.

**Optional Extra:** Features which would add extra visual appeal to the morphing results, but are not essential and will only be considered if extra time is found in implementation of the project.

Table 3.1 outlines all the requirements of the proposed morphing system.
### Table 3.1

<table>
<thead>
<tr>
<th>Requirement Reference Number</th>
<th>Description</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Object Representation</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Boundary Representation</td>
<td>Critical</td>
</tr>
<tr>
<td>1.2</td>
<td>Approximation of Boundary Objects</td>
<td></td>
</tr>
<tr>
<td>1.2.1</td>
<td>Octal Sphere Approximation</td>
<td>Critical</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Hubbard Sphere Approximation</td>
<td>Optional Extra</td>
</tr>
<tr>
<td>1</td>
<td>Volumetric Representation</td>
<td></td>
</tr>
<tr>
<td>1.2.1</td>
<td>Matrix Representation</td>
<td>Critical</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Marching Cubes Representation</td>
<td>Desired</td>
</tr>
<tr>
<td>2</td>
<td>Boundary Morph</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>Cross Fade</td>
<td>Critical</td>
</tr>
<tr>
<td>2.2</td>
<td>Boundary Projection</td>
<td>Desired</td>
</tr>
<tr>
<td>3</td>
<td>Volumetric Morph</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Intermediate Sphere</td>
<td>Critical</td>
</tr>
<tr>
<td>3.2</td>
<td>Explosion</td>
<td>Desired</td>
</tr>
<tr>
<td>3.3</td>
<td>No source object</td>
<td>Desired</td>
</tr>
<tr>
<td>3.4</td>
<td>Puddle</td>
<td>Desired</td>
</tr>
<tr>
<td>3.5</td>
<td>Vertex interpolation Acceleration and Dampening</td>
<td>Optional Extra</td>
</tr>
<tr>
<td>4</td>
<td>Showcase Game</td>
<td>Critical</td>
</tr>
</tbody>
</table>

3.8: Evaluation

Evaluation of the morphing process will be practical based. There is little in the way of numerical data that can be analysed. Therefore aspects of the morph will be evaluated visually against principles outlined as desirable for morphing, using a standard set of objects which aim to give a good representation of many varying types of object which the algorithm may face. The only numerical measure applicable to this project is the frame rate achieved during the morph when varying input variables, which will also be evaluated.
3.8.1: Principles for good morphing

(Gomes et al, 1998) outline ten principles which they conceive as giving good morphing results.

<table>
<thead>
<tr>
<th>Attributes Transformed</th>
<th>Graphical objects are made up from a set of attributes. The attributes present in the source object should be transformed into the attributes in the target object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topology Preserved</td>
<td>Given homeomorphic objects the topology of the source and target objects should be preserved during interpolation from one to the other.</td>
</tr>
<tr>
<td>Feature Preservation</td>
<td>Features of the source object should be transformed into features in the target. Vertices in the source should therefore be transformed into vertices in the target.</td>
</tr>
<tr>
<td>Rigidity Preservation</td>
<td>In brief, the rigidity preservation principle requires that some geometric properties of the source shape should be preserved during the transformation.</td>
</tr>
<tr>
<td>Smoothness Preservation</td>
<td>Smoothness preservation requires that the smooth regions (or boundaries) present in the source object should be ‘mapped’ onto the smooth boundaries of the target object.</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>The volume, areas, or parts of the source object should change monotonically, that is where possible these attributes should remain constant during interpolation.</td>
</tr>
<tr>
<td>Nonlinearity</td>
<td>Avoidance of linear interpolation of an object. A good example of where linear interpolation generates poor morphing transformations is during rotational morphing. A rotational morph is where the target shape is simply a rotation of the source; a linear interpolation of such an instance squashes the shape of two originals, whereas a better morph should preserve the shape of the source and simply apply a rotation.</td>
</tr>
<tr>
<td>Use of transformation groups</td>
<td>Refers to the situation where the target object can be achieved through a series of transformations rather than altering the geometry, in which case the transformation should take precedence over the latter.</td>
</tr>
<tr>
<td>Slow-in and slow-out</td>
<td>The interpolation of the vertices should accelerate at the beginning and decelerate at the end, giving a non-linear interpolation.</td>
</tr>
<tr>
<td>Avoiding morphing leakage</td>
<td>Applying a morph to a region where the object to be deformed resides, any deformation of the surrounding area is what is known as morph leakage. This phenomenon is more related to morphing in 2D space where there is no obvious outlines of shapes defined within an image.</td>
</tr>
</tbody>
</table>

Table 3.2 Principles for good morphing outlined by (Gomes et al, 1998)

Not all of the principles outlined in Table 3.2 will be appropriate for this new method of morphing. The sections that will be evaluated against are:

- Attributes Transformed.
- Rigidity Preservation.
- Monotonicity.
- Nonlinearity
- Slow-in and slow-out.

The remaining was thought to be not important or applicable to the proposed morphing algorithm, the justification for this will be argued. The interpolation procedure will not result in topology preservation or feature preservation being upheld as it is on the most part linear. Given two topologically different objects the morph will successfully morph between the two. However the topological features will not be preserved during the interpolation. The
knowledge of how the morph is expected to not uphold these features renders evaluation against them unnecessary, and so they will not be used as part of the evaluation procedure. Preservation of smoothness lies outside the scope of this paper, this paper does not attempt to create an algorithm for morphing textures, and therefore smooth boundary information is similarly not handled. This algorithm is designed for morphing from one object (source) to a different object (target), it isn’t however aimed to be used for morphing from one object to a translation of the same object; therefore use of transformation groups is not necessary and will also not be discussed in the evaluation procedure. Avoidance of morphing leakage does not translate from 2D morphing to 3D volumetric morphing, 3D objects are clearly defined within an object space, and so applying a morph just to the desired object is simple in comparison to a 2D image where the object of a morph has no clearly defined boundary.

3.8.2: Standard Set of Objects
Evaluation will be conducted using a standard set of polyhedra. The chosen polyhedra displayed in Table 3.3 were selected to test various aspects of the proposed morphing algorithm to ensure its performance with objects with varying features. The standard objects will all be morphed and observed for compliance with the previously outlined characteristics for good morphing.

<table>
<thead>
<tr>
<th>Model Reference Name</th>
<th>Number Of Vertices</th>
<th>Genus</th>
<th>Texture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzanne</td>
<td></td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Penguin</td>
<td></td>
<td>0</td>
<td>Applied</td>
</tr>
<tr>
<td>Torus</td>
<td></td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>Teapot</td>
<td></td>
<td>1</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 3.3 Standard objects to be used for evaluation of new morphing system.

Evaluation results will include testing of the following morphs:

1. Non-Textured genus 1 model -> textured genus 0 model (Teapot -> Penguin)
2. Textured genus 0 model -> non-textured genus 0 model (Penguin -> Suzanne)
3. Non-Textured genus 0 model -> non-Textured genus 1 model (Suzanne -> Torus)
4. Non-textured genus 1 model -> non-textured genus 1 model (Torus -> Teapot)

3.8.3: Frame rate measurements
The morphing algorithm proposed is to be a real time solution to morphing, therefore frame rate is a vital component of the morphing procedure. The frame rate will be evaluated while varying aspects of the morphing environment, these aspects of interest being:

- Source & Target model complexity, measured by the number of vertices.
- Voxel Grid resolution.
- Number of metaballs produced by the sphere approximation, morph stage 1.
Chapter 4: Design

The algorithm proposed can be thought of as four separate subprojects (Considering stages 3 and 5 to be the same). The design of the algorithm will be created as separate components which work together to create the overall desired morphing effect. The render loop state diagram which ties in all these component parts into an entire morph can be seen in Figure 4.1. By making the stages separate each stage can be worked on independently of the others. Each code update to a stage can simply be slotted back into the functioning system. As each stage is a complex process in its own right, it is likely that there is room for improvement in each of them which can be simply replaced as required at a later date. The entire coding design for the entire system can be seen in Figure 4.2, showing how each of the individual components will tie together in the programming code.

Figure 4.1 Morph procedure state diagram.
Morph Demo Application

The morphing demo is to have a simple layout to allow showcasing morphing of different objects. The application will have the ability to morph to four different models which are pre-specified. As the window manager being used is GLUT, complex forms are not feasible. Therefore the interface will be kept as simple as possible and handled by OpenGL. Figure 4.3 shows a design for the program. The models will be specified in the code for simplicity and selected through keyboard input at runtime. The set of four standard objects outlined in section 3.8.2 will be used. An alternative version will also incorporate more complex game models to illustrate the flexibility of the developed system.

![Figure 4.2: Simplified class diagram showing designed structure of code](image)

![Figure 4.3: Wireframe of demo morphing application.](image)

Showcase pairs game design

The Pairs game design will be kept simple and will consist of a board with a variable number of game pieces to be paired. Input is taken from the mouse and selected objects will be
calculated using picking. OpenAL will be used to play background music, and sounds when an object is clicked. The models used for morphing will be premade models taken from the game (DOOM3, 2004) as these will show the algorithm is capable of real world game application. Figure 4.4 shows a prototype of the game layout.

Figure 4.4 Prototype game board for showcase game.

1.) The background panels are created using a custom made panel class making image panels easy to create and place on a scene.
2.) Each game piece is selectable and will morph upon a user click.
3.) The game board is fully textured and can be rotated by dragging the mouse from side to side. This lets the user bring a game piece closer to the viewport but also adds another level of difficulty to the game.

Compute Unified Device Architecture (CUDA)
Metamorphosis between two objects using a volumetric approach is estimated to be very computationally expensive. Efficient Metaball calculation and representation is a major part of this project, carrying out the necessary calculations to attain a volumetric representation every frame is costly. After creation of functioning prototype Metaball code with skinning entirely on the CPU, the frame rate was too low. In order to gain raw computing power required to calculate multiple influence values the GPU was used.

CUDA formally introduced in 2006 is Nvidia’s general purpose parallel computing architecture. It was developed to process computational problems in a much quicker time than the traditional CPU. The number of threads offered by the architecture is massive. In the current latest GeForce 8 series architecture there are 128 thread processors available, each processor capable of managing up to 96 threads concurrently, giving a huge maximum of 12,288 threads. To utilise these threads Nvidia provided a hardware thread manager, meaning developers don’t have to write explicitly threaded code to harness their power. However, the programmer must keep in mind a threaded approach to application
development, analyse how best to divide application data to small chunks which can be distributed among thread processors.

The majority of the computation carried out on each frame cycle of the program will be carried out by a CUDA kernel. The basic overview of the process is as follows:

1) Clear previously calculated arrays. (GPU)
2) Calculate vertex influences of the voxel space. (GPU)
3) Calculate voxel values. (GPU)
4) Compact calculated voxel arrays. (GPU)
5) Draw skin (CPU)
6) Iterating to next frame position. (GPU)

This amount of dependency on GPU programming makes it a central component in the development of the morphing system.
Chapter 5: Implementation and Testing

Referring to the project implementation overview Figure 5.1 the implementation of the system will be discussed. The code design is focused around the Morph object (12), this is the central component which deals with all of the runtime morphing. The code is called in the render loop and so must be computationally inexpensive. To draw a new frame the morph object must first be passed a source and target morphModel(11&13). The morph models represent the source and target object which are present in the scene and handle the following:

- Loading vertex and texture from object wavefront model file format.
- Calculating vertex and face normals.
- Calculating (or loading if file present) sphere approximation data.

Keeping the morphModels external from the morph object gives greater flexibility in morphing between more than two objects, as it is possible to pass different source and target objects at any time to the morph object. The result is that only one morph object is required for all morphing objects in a scene cutting down on memory allocations, however if simultaneous morphs are required it would be more efficient to create two or more morph objects. The design of the code also allows the possibility to change the target currently being morphed to mid morph which may be desirable for some gaming applications which require a highly responsive system. In the following subsections the design of the major components will be outlined; they are split up into the initialisation stage and the runtime stage. It should be clear that efficiency was not a key component while implementing the initialisation stage; alternatively it was the driving factor in many of the design choices of the run time code.
Chapter 5: Implementation and Testing

Figure 5.1 Overall system components overview, displays the incorporation of C, C++, CU and Obj files.

5.1: Program Initialisation
At initialisation stage of the program, all the pre-processing required to produce real time morphing of models is carried out. These operations are not speed vital, therefore some areas which could have been made more efficient have been overlooked as non-vital to the project.

Overview the processes at initialisation:

- Loading models from file.
- Creating sphere approximation.
- Setup of marching cubes.

5.1.1: Model loader
The models are loaded into memory along with any textures they require. The textures are handled by a third party library SOIL. The data structure used allows easy calculation of normals, both face and vertex normals are calculated for each model and are both used in the morphing procedure. The model format used for ease was Object Wavefront due to its plain text nature.

5.1.2: Sphere Approximation
The octal approach described by (Liu et al, 1988) is straightforward and fast way to compute approximations of objects from spheres (see Section 2.1.3.1). It was implemented as the base level implementation of sphere approximations. Deciding if a point is internal to a polyhedral mesh was the largest challenge in the creation of sphere approximations.
5.1.2.1: Inclusion Testing
Deciding whether a sphere is internal to polyhedra can be done by ignoring the radius and testing whether the centre is inside the object. A simple inclusion testing algorithm is used; shoot a ray out to infinity in one direction. The ray is tested to see how many of the polygonal faces it intersects. If the number of intersections is odd then the test point is internal, similarly if the number of intersections is even the point is outside the polyhedra. Figure 5.2 illustrates this concept for the internal point P.

![Figure 5.2 Illustration of inclusion testing algorithm performed on the point P.](image)

5.1.2.2: Polygon Intersection Testing
In order to test if a ray intersects a polygon an inclusion testing algorithm was implemented. The code was kept simple to allow more focus on developing morphing. Referring to Figure 5.3 the intersection test will be explained.

For a point P in space, P is internal to the polygon if:
1. P lies in the plane of ABC and,
2. Area ABC = APB + ACP + CBP

This method is slow but simple and affective, and as already stressed speed is not an issue during this stage.

![Figure 5.3 Polygon inclusion testing on point P.](image)

5.1.3: Octree Generation
The octrees can be created to any specified resolution, giving the programmer flexibility in the number of spheres generated. Figure 5.4 shows the process for the creation of a octree of resolution 3.

![Figure 5.4 Generation of octree to resolution of 3.](image)
5.1.3.1: Sphere Approximation Test Results
The sphere tree generation code was tested on the standard objects (Section 2.8.2). It became immediately obvious that attaining a close approximation would require higher resolutions which produced exponentially higher numbers of spheres generated in the approximation. Referring to the generated sphere approximations of Suzanne Table 5.1, it can be seen that the number of generated spheres increases to a massive 2636 for higher resolutions. The advantage of more detail is eventually eclipsed by the memory overhead of the number of required spheres to calculate in the generation of the implicit surface. Therefore a revised octree approach was created and will be referred to as ‘active node calculation’.

<table>
<thead>
<tr>
<th>Model Name: Suzanne</th>
<th>Model Name: Suzanne</th>
<th>Model Name: Suzanne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution: 4</td>
<td>Resolution: 5</td>
<td>Resolution: 6</td>
</tr>
<tr>
<td>Number of Spheres: 44</td>
<td>Number of Spheres: 326</td>
<td>Number of Spheres: 2636</td>
</tr>
</tbody>
</table>

Table 5.1 Test results of sphere approximation code on the test model Suzanne at varying levels of resolution.

5.1.3.2: Active node calculation
Active node calculation combines the different levels of resolution generated by the oct tree approach in an attempt to bring down the number of generated spheres. The approach will be explained with aid of an example. The octree is first created the same as outlined in Figure 5.4. Then the active node calculation algorithm is run to attain which spheres should be included in the approximation. Figure 5.5 shows an example of how the active node calculation algorithm decides if a sphere is to be included.
Chapter 5: Implementation and Testing

Iterative generation of sphere approximation.

1. Starting from the lowest resolution octree.
2. Each corner of each cell is tested for inclusion in the mesh (see section 5.1.2.1 for inclusion testing).
3. If >4 corners are internal to the mesh (2 in 2D example) the cell is included in the approximation and all of the cells children are discarded. If however this is not the case the next resolution is evaluated for that cell.
4. This continues until either no further resolutions are available or all remaining children have been discarded.

Figure 5.5 Example of active node calculation to create efficient sphere approximations.

Sphere trees are then stored as an array, discarding previous hierarchy to increase efficiency and to free memory resources at run time. Test results for the active node calculation seen in Table 5.2 gave satisfactory decreases in the number of generated spheres in the approximation while still producing accurate representations.

<table>
<thead>
<tr>
<th>Model Name: Suzanne</th>
<th>Model Name: Suzanne</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original mesh</td>
<td>Resolution: 5</td>
</tr>
<tr>
<td></td>
<td>Number of Spheres: 74</td>
</tr>
</tbody>
</table>

Table 5.2 Finalised sphere approximation results for the test model suzanne.

The sphere approximation generation takes a significant amount of time to calculate as it depends on the CPU speed alone. Although this could be also handled by the GPU to give faster computation the generation of such sphere approximation is not central to the project, therefore the structure is treated as a full pre-process. Once the sphere approximation has been created it is saved to the file system for fast future reference, so greatly increasing initialization speed.
5.1.4: Creating Marching Cubes
Marching cubes are used to create the polyhedral surface from the volumetric data. Each marching cube is calculated at program initialisation and rendered to a display list to increase vertex pushing power. These display lists are simply placed in a one dimensional array with the key corresponding to its generated marching cubes key. The keys are generated by the program and referenced many times during render see Figure 5.6 which illustrates how these integer keys are generated.

![Voxel key assignments](image)

By using the 15 base cubes seen in Figure 5.7. All 256 combinations are created through rotational and mirror symmetry, the cubes are also scaled to the correct voxel size. Making these calculations at initialisation rather than runtime further reduces the number of calculations within the render loop.

![Base cubes](image)

5.1.4.1: Voxel Space
The object space is divided into a voxel matrix required for creating the implicit approximations. The voxel matrix resolution is variable, however high resolutions will have a reduced performance. Currently the voxel space is square; however there is no reason that this cannot become rectangular, and thus produce rectangular voxels; however time restrictions have stopped experimentation of this kind. The size of the voxel matrix is
currently user specified; a method for generating the size based on the models may take the form of using the largest bounding box and adding padding to it as the implicit approximations are always larger than the original boundary polyhedra.

5.2: Render Loop
During each pass of the render loop a lot of operations are carried out to calculate the morph. Therefore efficiency of these operations is vital to keep the frame rate real time resulting in most of these operations being carried out by the highly parallel GPU.

Overview of render loop operations:

- Calculation of vertex influences in voxel grid.
- Calculation of Voxel Values.
- Calculation and drawing of the volumetric skin.
- Interpolation of metaball centres.

5.2.1: Calculation of Vertex Influences
For each vertex in the voxel matrix, the influence of all the metaballs in the morph is calculated. Rather than using the computationally expensive equations outlined in the literary research of Blinn (Section 3.2.3), a simplified version was used. The simplified version removes the need to use the square root operation. The revised calculation algorithm is outlined in Algorithm 5.1.

<table>
<thead>
<tr>
<th>Calculation of Vertex influence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For each</strong> metaBall</td>
</tr>
<tr>
<td>XDistance = metaBallPositionX – vertexPositionX</td>
</tr>
<tr>
<td>YDistance = metaBallPositionY – vertexPositionY</td>
</tr>
<tr>
<td>ZDistance = metaBallPositionZ – vertexPositionZ</td>
</tr>
<tr>
<td>distance = (xDistance * xDistance) + (yDistance * yDistance) + (zDistance * zDistance)</td>
</tr>
<tr>
<td>calculatedInfluence = (blobConstant * metaballMagnitude) / calculatedDistance</td>
</tr>
</tbody>
</table>

The blobConstant affects how attractive the metaballs are to one another, a greater constant increases the overall size of the mesh generated to the point that it loses all detail of the sphere approximation. Figure 5.8 shows test results of increasing the blob constant on the implicit surface for the test model Suzanne.
Chapter 5: Implementation and Testing

5.2.2: CUDA Implementation

All the major run time calculations were handled by parallel computing:

- Vertex Influence Calculation.
- Voxel Calculations
- Metaball Interpolation.

Example of how the threads were setup to give highly parallel performance.

The vertex influence calculation kernel allocates a thread to each vertex in the matrix. As the maximum number of threads per block is 512 the vertex grid would be limited to 8x8x8. To overcome this additional thread blocks are incorporated. Unfortunately the dimension of these are limited to 2 dimensions, so in order to attain a structure as in Figure 5.9 the width dimension is extended to store the depth information and manipulated at run time. Unfortunately the method used to achieve this isn’t efficient, so the application attempts to use as many threads available in each block to avoid use of the work around. The `threadsPerBlock` and `numThreadBlocks` are generated at program launch, calculated from the desired resolution of the vector grid.

5.2.3: Performance Issues

Achieving high resolution implicit meshes suffered from lower frame rates. This was partly down to the exponential nature of the number of required threads when increasing the
The influence calculation kernel is relatively computationally expensive compared to the other kernels; this is due to a sequential calculation within the thread. The sequential calculation is a loop of all the metaballs in the scene, this can be a large number, and for example the standard torus object generates 1032. This number doesn’t seem like a large loop; however the parallel nature of the threads does not lend itself well to this sort of computation. To reduce the bottleneck two extra stages were produced. An overview of the new approaches can be seen in Table 4.3 and the following subsections explain the ideas in greater detail.

<table>
<thead>
<tr>
<th>Independent Skin Resolution</th>
<th>Separating the vertex influence sample resolution and the skin generating resolution and interpolating between them. Gives higher resolution meshes using lower less accurate influence samples.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Calculation Filtering</td>
<td>Cuts down the amount of metaballs which are considered when calculating vertex influences by ignoring metaballs and vertices which are unlikely to have a large effect on the value.</td>
</tr>
</tbody>
</table>

Table 4.3 Computation cutting algorithms

5.2.3.1: Independent Skin Resolution
Sacrificing accuracy for speed is a common choice made in computer graphics. The morphing algorithm is aimed to be real time, therefore the decision to create a method which would produce higher resolution meshes using lower detail vertex calculations were considered the correct direction to take. Figure 5.10 shows the basic stages required in calculating vertex influences given a lower resolution influence voxel space.
Chapter 5: Implementation and Testing

5.2.3.2: Vertex Calculation Filtering

Every influence vertex value is calculated every pass of the render loop. However, there is a great possibility that these influence values will be insignificant in the implicit surface generation. This technique aims to cut out such vertices from the calculation loop. The vertex filtering algorithm will be explained referring to Figure 5.11. By ordering the metaballs (displayed in green) by distance from centre it is possible to attain the nearest and furthest metaballs (displayed in black). A threshold is then generated around these positions (Indicated by T). Any vertices which lie outside of this area of influence are ignored during vertex influence calculations (indicated by the darker areas).

5.2.4: Calculating Voxel Influences

The vertex influences matrix data is combined into voxel values. By assigning values to each vertex of the voxels a unique key can be created for each combination of voxel values which correspond to the marching cubes keys, see Algorithm 5.2 which shows how this key is calculated per voxel. The threshold value is variable and has much the same effect on the final generated structure as the blob constant.
Chapter 5: Implementation and Testing

5.2.5: Removal of unused voxels
To prevent examining empty voxels when generating the polygonal skin, the unused voxels are removed from the calculated voxel arrays. This is important as the skin will be called by the CPU, it requires a dynamic array and memory location to be carried out at run time.

5.2.6: Rendering of Implicit Surface
The skin is drawn using Marching Cubes. On render the compacted calculated voxel calculations are examined and the correct marching cube is selected from the array and called to be rendered in the correct position (Algorithm 5.3).

![Image](image.png)

Figure 5.12 Example marching cube generation by integer key value.

5.2.7: Correspondence of metaballs
The correspondence between metaballs is created very simply as a one to one relationship between the source and target object arrays. Any additional spaces are padded with pseudo randomly chosen points in the target object to ensure that every ball in the source corresponds to a ball in the target. Similarly given a target array of smaller size than the source, the remaining balls are corresponded to any random ball in the target. This approach
to correspondence leaves possible updatable aspect of the morphing system which is discussed in more detail later in Chapter 6.

5.2.8: Interpolation
Euclidean linear interpolation is implemented in the morphing system, taking advantage of the avoidance of edge collision issues discussed earlier. This simple interpolation system gives consistent predictable results. Vertex interpolation acceleration and dampening specified as an optional extra requirement (requirement number 3.6.2) was not implemented. This effect is still considered to give more fluid motion to the overall motion of the morph and would be a focus of future work on the system.

5.2.9: Boundary Morph technique
An additional stage to the morphing procedure was added that was not specified in the requirements stage. This stage was added to enhance the effect of the boundary to volumetric morph (Morph stage 3&5). The original effect looked too much like one object taking over another, the new uses simple texture cross fading technique and sphere mapping to distract and further convince an observer into thinking the boundary and volumetric objects are the same object. Figure 5.13 shows the source and target objects of this added morphing stage. The source is vertex shaded and displays any texture associated with the model file. The target model uses face normals and uses environment mapping just as the volumetric approximation of the following stage so giving a more convincing transition between the two. The effect is achieved by simple OpenGL blending. This additional morph state was included and can be seen in the morph state diagram Figure 5.14 as additional stages 1&6.

![Figure 5.13](source/target_objects.png)

**Figure 5.13** Additional morph stage to remove textured model (left) and replace with sphere mapped model (right).

5.2.10: Intermediate objects and Meta objects
The morph state procedure was altered to incorporate intermediate objects as standard. The implemented morphing procedure allows for any amount of morph models to be specified in a vector, with each full morph becoming a morph between all objects passed in the vector, this additional loop can also be seen in the revised morph state diagram Figure 5.14. An extra flag was added to the morph models to specify them as meta-objects. The idea behind this was to allow a morph to the meta-approximation of a shape without a full morph to the boundary occurring, which gives the ability to specify intermediate shapes. The intermediate shapes allow for the extra morphing effects specified in Section 3.4.1 without explicitly hard coding them.
Figure 5.14 Revised morphing system state diagram to incorporate additional steps and recursion.
5.2.11: Game implementation
The game was implemented using the prototype code; the noticeable changes were simply texture and model selection. The morphing algorithm was simply placed into the code and wrapped within the game piece class. Audio was supplied by the OpenAL framework and plays simple sound files on morphing, and background music throughout the game. During implementation a design change was taken due to unforeseeable performance issues being encountered using the GL_Select method for object selection. This performance issue seems to be a graphics card issue. The solution was to strip the use of the mouse in the game and instead use the keyboard for all user input. Aside from these small changes the game was implemented just as outlined in the specification.
Chapter 6: Results and Discussion

The morphing algorithm results will be discussed with reference to the evaluation principles outlined in the Section 3.8.1 and the original requirements Table 3.1. The morphing procedure was fully implemented allowing each aspect of the morphing procedure to be presented in this section and critically evaluated. The results have been split into sections which concentrate on the specific section of the morph:

- Sphere Approximation,
- Implicit Surface Generation
- Morphing In-Betweens Generation

The algorithm implementation is then evaluated against the original. The standard models seen in Figure 6.1 were used in testing each stage of the morphing procedure. Finally, algorithm results for real world game models are presented and discussed.

<table>
<thead>
<tr>
<th>Model Reference Name: Suzanne</th>
<th>Model Reference Name: Penguin</th>
<th>Model Reference Name: Torus</th>
<th>Model Reference Name: Teapot</th>
</tr>
</thead>
</table>

Table 6.1 Sample objects for evaluation.

6.1: Sphere Approximation Results (Requirement reference 1.2.1)

The sphere approximation section of the algorithm produced results seen in Figure 6.2. The sphere sizes in the images are not representative of the influence each sphere possesses as it isn’t easy to translate this data to solid sphere size due to their sizes being optimised for implicit surface generation; instead they are simply representation of the positioning of the spheres. The resolution tested was 5 as this gave a good trade-off between resolution and number of produced spheres. All standard polyhedra produced accurate sphere approximations; however there was large variation in the number of spheres generated. The Torus produced a large 1032 spheres due to its very round and thin structure, this large number of spheres impacted the frame rate for morphs using it.
Chapter 6: Results and Discussion

Table 6.2 Test results of sphere approximation algorithms on test objects.

<table>
<thead>
<tr>
<th>Mesh Name: Suzanne</th>
<th>Mesh Name: Penguin</th>
<th>Mesh Name: Torus</th>
<th>Mesh Name: Teapot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct Tree Resolution: 5</td>
<td>Oct Tree Resolution: 5</td>
<td>Oct Tree Resolution: 5</td>
<td>Oct Tree Resolution: 5</td>
</tr>
<tr>
<td>Number Of Balls: 182</td>
<td>Number Of Balls: 221</td>
<td>Number Of Balls: 1032</td>
<td>Number Of Balls: 198</td>
</tr>
</tbody>
</table>

6.2: Implicit Surface Generation Results (Requirement reference 1.2.2)
Figure 6.3 shows the implicit approximation of the standard polyhedra. It can be seen that each of the generated approximations are identifiable as an approximation of the original. The number of voxels required to represent the surfaces are largely similar and small when compared to the number of possible voxels (for resolution 27 the maximum number of voxels in surface is 19683).

Table 6.3 Test results of implicit surface generation on test objects.

<table>
<thead>
<tr>
<th>Mesh Name: Suzanne</th>
<th>Mesh Name: Penguin</th>
<th>Mesh Name: Torus</th>
<th>Mesh Name: Teapot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Of Voxels: 966</td>
<td>Number Of Voxels: 1821</td>
<td>Number Of Voxels: 1104</td>
<td>Number Of Voxels: 961</td>
</tr>
</tbody>
</table>

The surfaces of the volumetric structures produced by the Marching Cubes algorithm appear angular. The angular nature is due to the use of face normals, a decision made so display lists could be used for simple implicit mesh generation. Due to the static nature of these display lists it is not possible to calculate dynamic vertex normals. Another downfall of the display list method is that it is called by the CPU, in the morphing code a display list is called for every voxel calculated as being on the surface, this is a computational bottleneck which would be better avoided. Future work on this approach to morphing could implement VBOs rather than display lists. A display list is a vertex array which resides in GPU memory. VBOs can be mapped directly to the GPU calculated voxel calculations using CUDA and therefore removing
the CPU call bottleneck. Unlike Display Lists VBOs can be altered after creation, therefore it would be possible to calculate vertex normals which would result in a much smoother looking implicit surface being generated. The normals would be calculated much the same way as the influences of the metaballs on the vertices in the voxel matrix are. Smoother results would resemble a more fluid object much closer to water.

6.3: Morphing In-Betweens Generation Results (Requirements reference 2.1&3)
The developed morphing solution was able to produce morphing results for all standard objects. The following subsections show the results of the test morphs outlined in Section 3.8.2, and discusses their success based on the principles for good morphing also outlined in Section 3.8.1.

6.3.1: Morph Test 1 (Teapot -> Penguin)

Discussion of Test morph 1 seen in Figure 6.1. The morph successfully transitions from a genus 1 object (teapot) to a genus 0 object (penguin). Though the morph is an interpolation between genus values 0 and 1, a model with any genus will morph equally successfully. The source model in this instance is a non-textured model and morphs correctly into a fully textured target. Attribute transformation does not apply to this model as no obvious corresponding features are present between the source and target. The teapot model is a much higher resolution mesh than the penguin however this does not affect the overall morph produced.
6.3.2: Morph Test 2 (Penguin -> Suzanne)

Discussion of Test morph 2 seen in Figure 6.2. The morph shows a successful transition between objects of the same genus, 0 in this case. The resolution chosen appears to be more suited to the larger Penguin model than Suzanne as when the volumetric in-betweens are closer to the size of the target they appear quite low resolution. This low resolution can be corrected by increasing the resolution variables. These meshes do have corresponding attributes; both possessing eyes and noses. The in-betweens generated in the morph don’t use this information in the correspondence and so these features are not transformed into each other. The texture information of the source model is successfully removed before the morph takes place. Incorporation of texture morphing is a possible direction for future work on this technique of volumetric morphing. If the texture information could be also morphed during the volumetric morph the switch between boundary and volumetric objects would be less obvious, which is one of the main issues in creation of this process. Currently the volumetric object is textured by a sphere map, chosen as a simple texture to apply to the process. A more advanced version would be to use a cube map which is generated by the projection of the texture components of the source and the target. The two generated cube map textures would be simply faded into one another at morph time and used as the texture for the generated volumetric structure.

6.3.3: Morph Test 3 (Suzanne -> Torus)
Discussion of Test morph 3 seen in Figure 6.3. Morphing from a genus 0 to an extreme example of a genus 1 model is successfully carried out. The volumetric approximation appears to eat a hole in the centre as the balls part to their final positions. There was a slight fps decrease when using the torus due to the large amount of spheres in the approximation.

6.3.4: Morph Test 4 (Torus -> Teapot)

![Figure 6.4 Generated in-betweens morphing from Torus to Teapot (Morph Test 4).]

Discussion of Test morph 4 seen in Figure 6.4. The morph from torus to teapot shows an example of two higher genus objects morphing into one another. Though these objects do produce a successful morph, the generated in-betweens appear to lose all shape of the source and target models mid morph.

<table>
<thead>
<tr>
<th>Test Morph</th>
<th>Attributes Transformed</th>
<th>Rigidity Preservation</th>
<th>Monotonicity</th>
<th>Nonlinearity</th>
<th>Slow-in and Slow-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morph Test 1</td>
<td>NA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Morph Test 2</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Morph Test 3</td>
<td>NA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Morph Test 4</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 6.4 Overview of test morphs adherence to specified evaluation criteria.

Overall these test morphs show that on the most part a successful algorithm was created which does create satisfactory morphing results. However it is evident that the morphs produced failed to meet the requirement of attribute preservation summarised in Table 6.4. This is the matching of a source attribute (such as a nose) to a corresponding attribute on the target model. This information was not directly available to the program with the techniques used. Without human intervention of tagging each of the corresponding metaball spheres from source to target there is currently no information that would hint which area should correspond to which area. A simple method could be used by upgrading the mesh model loader to import more advanced model formats which incorporate bones. 3D models which incorporate a skeleton often follow a naming convention for the bones. This naming convention is used for easier interpretation by humans and so is human readable, this information could be easily used to match specific areas of spheres in a source mesh with specific areas in the target mesh. Alternatively a move from the octree approach for object sphere approximation to that outlined by (Hubbard, 1996) could offer such information. Hubbard sphere trees offer the additional feature of medial axis generation, if the generated
medial axis of the source and target meshes could be corresponded, this correspondence could be used for matching the spheres which created it from source to target in a more intelligent manner. The Hubbard approach could also lead to quicker interpolation due to reducing the number of spheres generated for each mesh approximation.

6.4: Intermediate Mesh Morphs (Requirements reference 3.1-3.4)

Intermediate mesh morphs are possible using the new morph state structure of Figure 5.14. Figure 6.5 shows a morph between torus->Suzanne->Penguin. The full boundary of Suzanne is never formed as it is flagged as a meta-object. This morph can be carried out with any number of intermediate objects.

Morphing to a puddle is simply a case of supplying the morph component with three models, one for the source, one for the puddle (flagged as a meta-structure) and finally a target model. A puddle morph can be seen in Figure 6.6, in this example the puddle was created as a simple cylindrical object. The developed morph system makes it very simple to include such complex multiple stage morphs.
6.5: Original Specification Comparison
Figure 6.7 shows an overview of the processes required in creating a morph with example results for each stage. The developed algorithm stayed true to the original specification in the requirements section see Figure 3.1. All critical and desired features of the original specification were implemented in the project; one feature not to be implemented was interpolation acceleration and damping. This noncritical feature is still thought to have a positive impact on the overall motion of the morph and would still be a focus of future work.

![Diagram showing processes required in creating a morph]

Figure 6.7 Direct comparison to original specification diagram.

6.6: Testing on more complex Game Models
To ensure that the morphing algorithm was not dependant on the test models to which it was created, further tests were carried out on real world game models. The game models were taken from the game (Doom3, 2004) and represent models which the morphing algorithm must be able to process if it were to be incorporated into a real world game.
Table 6.5 Example implicit surfaces generated for more intricate game models.

The volumetric approximations of the game models tested in Table 6.5 do adequately approximate the original boundary. Even a Trite model which exhibits very thin features produced an adequate approximation of the original. This accuracy can also be increased by increasing the octree resolution.

Figure 6.8 Generated in-betweens morphing between two more intricate game models.
Testing the morphing system on more complex game models gave satisfactory results, meaning that a morph was successfully generated for each of the game models tested. Each stage of the morphing process successfully ran with the more complex game models. However it can be seen in Figure 6.8 and the previous morphs that correspondence of attributes is currently lacking which creates confusing in-between structures which do not appear to have a great relevance to either the source or target models.

6.7: Performance Results

All performance results were conducted on the development computer system the specifications of which can be seen in Table 6.6.

<table>
<thead>
<tr>
<th>Processor</th>
<th>AMD Athlon 64 X2 Dual Core Processor 2.00GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ram</td>
<td>2.00GB</td>
</tr>
<tr>
<td>Graphics Card</td>
<td>BFG GeForce 9800 GT (1GB)</td>
</tr>
<tr>
<td><strong>CUDA Specific:</strong></td>
<td></td>
</tr>
<tr>
<td>Number of multiprocessors</td>
<td>14</td>
</tr>
<tr>
<td>Number of cores</td>
<td>112</td>
</tr>
<tr>
<td>Operating system</td>
<td>Microsoft Windows 7 (64-Bit)</td>
</tr>
</tbody>
</table>

Table 6.6 Development computer system specifications.

Although the code was written with efficiency in mind, realising a real time morphing algorithm of this nature wouldn’t be possible without CUDA. The frames per second increase over the CPU can be seen in GRAPH 6.1. The test was conducted on the vertex influence calculation kernel, displaying no geometry. The code for two different methods was not the same as each were optimised to achieve the best results on the corresponding platform. Utilising CUDA clearly gave an extra push to allow higher resolution morphing without suffering frame rate loss. The framework was also manageable and relatively simple to use. Testing the entire morphing algorithm on the CPU was not possible as development of equivalent CPU morphing functions was not carried out after this test showed the CPU to be unsuitable. However Graph 6.1 shows that the CUDA code also eventually starts to drop frames, the performance of the entire morphing system is presented in Table 6.7. There are many factors which contribute to the frame rate produced making it hard to predict produced performance before test. The settings which give satisfactory frame rates are indicated in green.
Graph 6.1 Comparison of CPU and GPU processing speeds of vertex influence kernel.

<table>
<thead>
<tr>
<th>Skin Sample Resolution</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Influence Sample Resolution</td>
<td>10</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>17</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>29</td>
<td>20</td>
<td>13</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>13</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.7 Performance measurements of morphing algorithm (frames per second)

6.8: Morphing Showcase Game (Requirement reference 4)
The showcase Pairs game successfully runs at 30fps and incorporates 10 different possible models all taken from the game Doom3. Figure 6.9 shows a screen capture of the game running with a morph currently running in the foreground. For a small game there are a lot of resources required, on the development computer it had a noticeable slowdown if more than one morph were to be displayed at the same time. To avoid this slow down the player was limited to selecting one object at a time.
Figure 6.9 Screenshot of the developed morphing showcase Pairs game
Chapter 7: Conclusion

7.1: Overview
The aim of this paper was to design and implement a new approach for morphing polyhedral objects. After research in the area the aim was further refined to prove that a boundary-volumetric morph would be an interesting way to morph between two or more polyhedra. The new proposed system incorporated volumetric object approximation in order to create a structure which can be interpolated easily without edge intersection or issues of previous methods.

7.2: Implementation
Overall the system was successful in producing real time boundary-volumetric morphs between objects with largely varying features. The technique was therefore proven to be an effective alternative to the existing techniques for morphing. An additional feature of incorporating any number of intermediate objects mid morph allows for the new technique to produce more interesting morphs than just straight interpolation from a source to a target, such as morphing to a puddle. Such effects may be desirable in gaming applications which may require a more interesting effect.

The morphing system incorporated GPU processing to offer real time morphing calculations which would integrate easily into a game. The GPU was programmed using the CUDA toolkit and played large part in the realisation of this new morphing system. To illustrate the opportunities of using this technology a showcase game of Pairs was developed which incorporated the boundary-volumetric morph system directly. The game included user interaction and all animations were generated while keeping a desirable frame rate.

7.3: Future work
The main focus of future development of this morphing technique would be to include smoother interpolation of volumetric objects. Currently the volumetric object is created with face normals producing an angular look to the surface. The addition of Vertex Buffer Objects would increase speed and allow for calculation of vertex normals to achieve smoother surfaces.

The introduction of attribute preservation during a morph would improve the correspondence algorithm and produce a more intelligent morph. Such attribute preservation requires labelling of key areas on the source and target objects, more advanced sphere approximation algorithms such as the one outlined in the paper by (Hubbard, 1996) could offer such label information.
References


DOOM3 [Computer Game]. (2004), Activision. ID Software


*Harry Potter and the Philosopher’s stone* [Motion Picture]. (2001) Columbus, Chris.


SOIL – Simple OpenGL Image Library. Available at: http://www.lonesock.net/soil.html
[Accessed: 30th February 2010]

*Spiderman 3* [Motion Picture]. (2007) Raimi, Sam.


