Modern Science and Art University
Faculty of Computer Science

Graduation Project Documentation

3D Smooth Blending for subdivision surfaces

Semester (65)

Spring 2017/2018

Submitted by:

Name: Yousef Hesham Samir

ID: 153137

Supervised by:

Dr. Ahmed Farouk

Graduation Project Spring 2018

Prepared by

Yousef Hesham Samir

Supervisor

Dr. Ahmed Farouk

Abstract

With rapidly growing interest in the world of 3D models, many techniques are developed in order to attain better and more realistic results. This project is a program that builds and views 3D models, and convert them into subdivision surfaces, smoothing them after each iteration. Subdivision surfaces are initially a control mesh that has been refined through recursive subdivision, each iteration is always defined by the previous iteration's vertices

Then the program works on blending two or more objects together while maintaining the smoothness of both objects. Blending is a method for combining freeform objects represented with subdivision surfaces. The technique investigates the deformation of the surfaces and at the same time maintain the smoothness of the blending area that will connect the two objects. The two objects to be blended have a blend region in between. The geometric properties of the surfaces in the neighboring areas of the blend area is subdivided adaptively and the border curves of the blend area between the two objects are smoothed to minimize possible deformation in the blending surface. Using a blend curve, we can connect the two objects.

Table of Contents

Table of Figures

Chapter 1: Introduction

Chapter 1: Introduction

1.1: Rationale

For decades, 3D modeling and blending has been a tremendously crucial branch of the 3D object creation and manipulation. Nearly in every game, application or movie we see some sort of blending technique to give a certain effect or action. Blending freeform objects modeled with subdivision surfaces can be seen around is all the time nowadays.

1.2: Aim

The aim of this project is to develop a software than can load/view 3D objects. Then apply subdivision iterations on these objects to create smoother meshes with higher vertex point density. As well as blending these freeform objects together while maintaining the overall smoothness of the objects.

1.3: Motivation

The field of computer vision is continuously growing over the years. The use of system based on computer has increased dramatically in the past decade. Film industries use computer vision and blending in animated movies, action movies and others in order to manipulate the objects/environments of the film. Surgeons use systems based on computer vision to display predicted results of surgeries. Numerous mobile applications rely on the developments made in the field of computer vision. Advances in automobile engineering are based on the use of computer vision and many more. Nearly every profession or filed makes use of advances in computer vision. The ongoing growth in use of the field of computer vision is an inspiration to develop the presented software.

Chapter 2: Background and Related Work

Chapter 2: Background and Related Work

2.1: What is subdivision? The Fundamentals

Subdivision, is simply defining a smoothed surface as a finite surface resulting from a process that uses a control mesh and repeatedly refining it in multiple levels and constantly adding new vertices, faces and edges.

2.2: Chaikins subdivision algorithm

In Fig. 1, we can see a 4 edge curve (closed) that has been refined 3 times using the corner cutting method, which is a technique used in Chaikins subdivision algorithm. Each control vertex of the curve in the iterative mesh is calculated as a percentage mixture of previous bordering vertices. To that end, the new mesh changes to a smoother curve, which is widely known as uniform quadratic B Spline curve.

† In Chaikins subdivision scheme, the topological rules are shown in Fig. 2, which is widely called as corner cut technique. For each control/previous vertex Vi, we cut off the corner curve by adding two vertices, which create a new edge that connects the two recently added vertices, the overall lengths of all previous edges will be reduced.

† In Chaikins subdivision scheme, the geometric rules are located by the following equation. The recently inserted vertices and they are calculated as a linear combination of old bordering vertices.

$$v'_{2i} = \tfrac{1}{4} v_{i-1} + \tfrac{3}{4} v_i \qquad \text{(1a)}$$

$$v'_{2i+1} = \tfrac{3}{4} v_i + \tfrac{1}{4} v_{i+1} \qquad \text{(1b)}$$

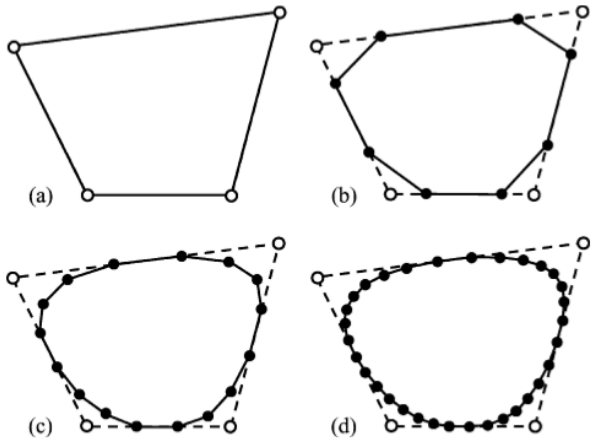


Fig. 1. Subdivision through Chaikin's corner cutting algorithm: (a) the initial control mesh; (b)–(d) control meshes after one, two and three subdivisions, respectively. [4]

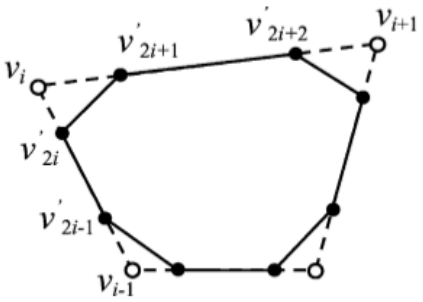


Fig. 2. Topological rules for Chinkin's subdivision.[4]

2.3: Doo–Sabin subdivision surfaces

Doo-Sabins subdivision surface is considered a general concept of one of Chaikins subdivision's versions. In a regular rectangular control mesh, uniform biquadratic B Spline surfaces are produced. Here are the set of topological and geometric rules for Doo-Sabin subdivision surfaces.

In Fig. 3, we can see the topological rules of Doo–Sabin subdivision surface. For each face that contains n vertices, we will insert n new vertices and are calculated by following the geometric rules that will be discussed later. The new iterative mesh is built by connecting the related vertices to form F-faces, E-faces and V-faces as shown in Fig. 3. Foe each previous face, a new F-face is built using all recently added vertices of the parallel face. For each previous edge, new E-face would also be built by connecting the four recently added vertices of the previous edge. For each previous vertex, a V-face can be built by connecting recently added vertices.

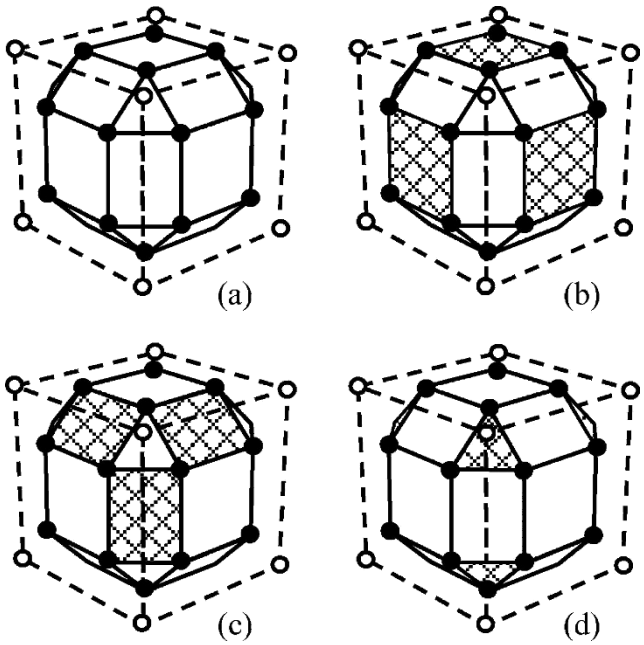


Fig 3. Doo–Sabin subdivision rules. [7]

2.4: Different subdivision schemes and a brief overview

As mentioned before, any subdivision scheme is known to use a set of geometric and topological rules, these rules are used for mesh improvement in general. Topological rules usually describe how a control mesh is converted to an iterative mesh. The common operations of topological rules may include adding new vertices into faces or edges data structure, renewal of previous vertices, edges, connecting recently inserted/updated vertices (along previous vertices if required), and removing previous unwanted vertices, faces or edges, all these rules are used depending on the style of a subdivision surface we're dealing with. To calculate the exact coordinates of the new iterative vertices we use geometric rules. Some important properties must be considered when trying to design geometric rules for subdivision, finite support with small masks, surface behavior itself, affine invariance, and symmetry. in Fig. 4, some basic topological rules are summarized for subdividing quadrilateral and triangle objects.

In some schemes we can find that they incorporate more than one of the topological rules mentioned in Fig. 4. In Fig. 5, a loop subdivision surface is using the one to four splitting topological rule for triangular meshes. Fig. 6 shows a model of a pipe and pistol, made using Catmull-Clarck subdivision and Doo-Sabin subdivision.

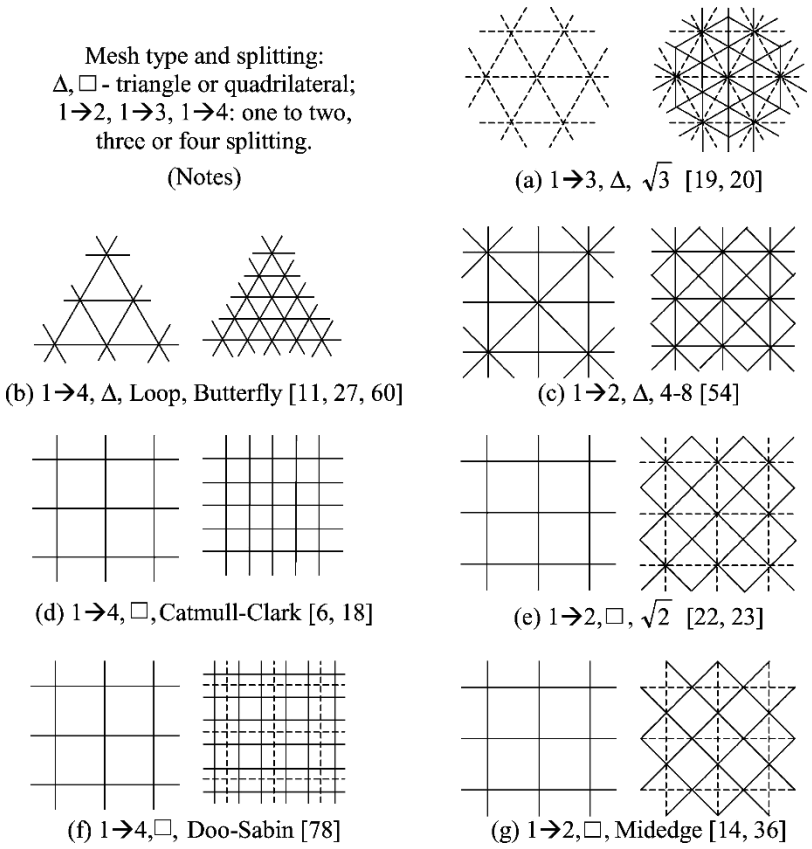


Fig. 4. Several important topological subdivision rules. [7]

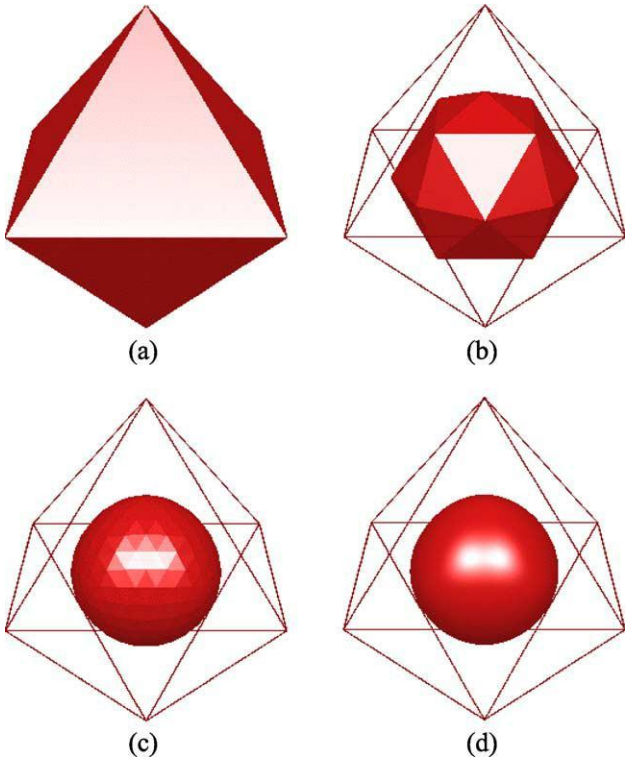


Fig. 5. Example: Loop subdivision scheme (a) starting object mesh (b) object mesh after one iteration (c) object mesh after three iterations (d) the limit surface. [7]

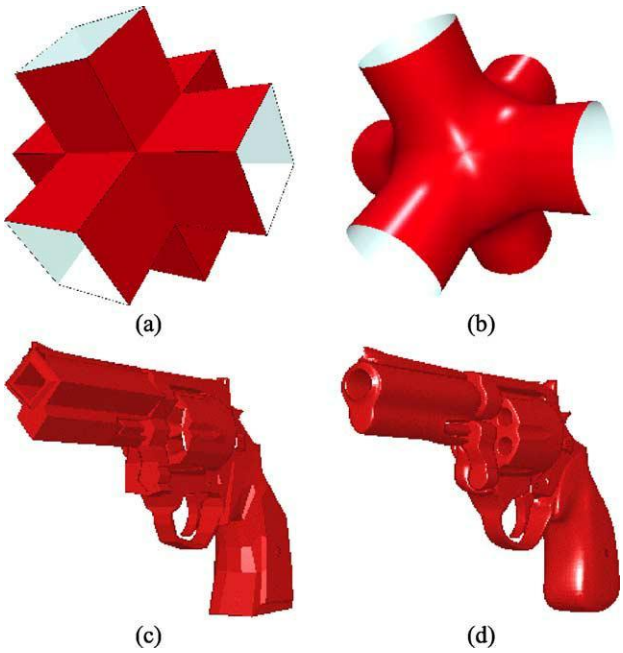


Fig. 6. Example: pipe model and a pistol model. [7]

2.5: The Half-Edge Representation Structure

What is the Half-Edge Data Structure? It is a common method used to demonstrate a polygon based mesh, using a shared list of points and a list saving polygons saving direct pointers for its vertices. This description is not only appropriate in most situations but also reaches maximum productivity; however, in some other purposes it may be ineffectual.

The process of mesh refinement sometimes needs breaking down an edge to a single vertex, which will require removing all the faces neighboring the edge as well as updating the faces that share these vertices. Which demands us to detect adjacency relationships between faces and vertices of the mesh. These operations can most certainly be performed on the simple mesh exemplification mentioned before, but it will definitely be of high cost, many of these operations will need a long search over the whole list of vertices or faces, sometimes both will be needed.

Some adjacency queries on a simple polygon mesh may have:

1) What are the faces that contain this vertex?
2) What are the edges that contain this vertex?
3) What are the faces that neighbor this edge?
4) What are the edges that neighbor this face?
5) What are the faces that are neighbors to this face?

In order to perform these kinds of adjacency queries in an efficient way, the (b-reps) representation method have been developed which is a more sophisticated way to directly model the vertices, faces and edges of the mesh with extra information saved inside. The winged-edge data structure is one of the most popular types of representations, in which edges are enhanced with pointers to the vertices creating that edge, and the two faces neighboring that edge, and also four pointers to the edges which come out from the end points. With this representation structure, we can locate which faces or vertices neighbor a specific edge in fixed time, but, some other kinds of queries may need more costly processing time.

The half-edge representation is built to be a very advanced version of b-rep, that grant all the queries mentioned before to be performed in constant time. Furthermore, all adjacency information stored inside the faces, edges and vertices stay with fixed space, which means no changing arrays will be used while also providing reasonable compatibility.

With such characteristics that the half-edge data structure carry, it has been an outstanding option for many programs, however it's limited to representing only manifold surfaces, which in some extreme cases can be difficult or impossible to implement.

Structure

From the name, the half-edge representation is named this way simply because we don’t need to save the whole edges of a mesh, we only save half-edges, when we split an edge midway, it created two half edges, each called a half of an edge, the two half edged together are called a pair. All half-edges must be directional, and each pair of edges is facing a different direction.

Fig.7 illustrates a part of a representation in a triangle mesh using half-edge data structure. Yellow dots represent vertices of the object, blue lines are half-edges. while yellow arrows demonstrate pointers.

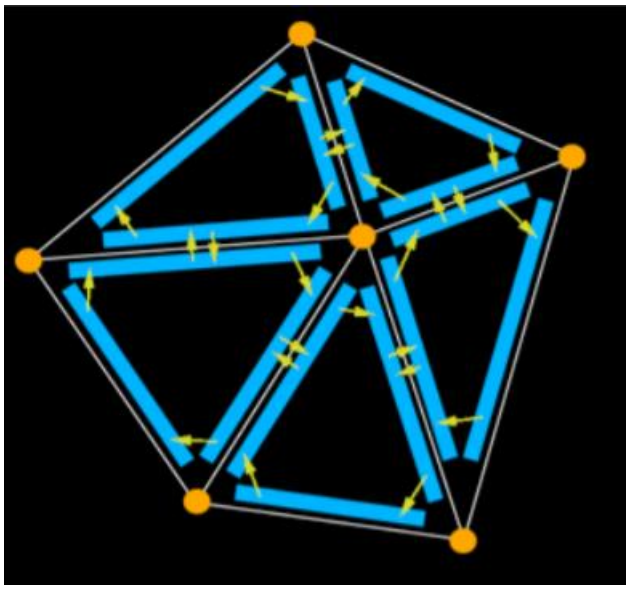


Fig. 7. Representation of a triangle mesh using half-edge data structure. [8]

In Fig.7, all half-edges that neighbor a particular face must form a linked list in a shape of a circle around the face's perimeter. This circle will be directed either clockwise or counter-clockwise as long as it’s around a face, and as long as the same orientations will be used all along. Each of these half-edges included within the circle stores one pointer to the next border, the vertex at the end, and a pointer to its pair.

Each vertex stored within the half-edge representation saves an x, y and z position value, in addition to a pointer pointing to one of the half-edges only, which then may use this vertex as it's initial starting point in order to create the circular linked list we mentioned before. At a given vertex, we will always have more than one half-edge to choose form, however only one is needed and it would not matter which one is chosen.

For a very basic form of a half edge representation, each face will need to save a pointer to only one half edge neighboring to it, while being in a more complex environment we would most likely save information on normal vectors, colors and textures as well. Also In the faces, any half edge pointer in the face is very identical to a vertex structure, however in a face there are several half-edges neighboring each of the faces, but in our case we can only need one of them to save and it wouldn't matter which one we choose.

Adjacency Queries

For all adjacency queries, the answers are directly stored in the data structure of the vertices, faces, and edges.

Iterating over the half-edges bordering a face is a slightly more complex example, the half edges around a certain face form a circular list, and the face data structure stores a pointer to only one of these half-edges.

Similarly, we might want to iterate over the faces or edges which are bordering to a certain vertex, going back to Fig. 7, you can see that not only do we have circular linked listed around the borders of each face, but also we can form loops around the vertices.

2.6: Libraries Available

2.6.1 OpenGL

Open Graphics Library is an API normally used to associate with GPU, and it has been used for rendering 2D and 3D designs and meshes.

2.6.2 DirectX

Direct3D (the 3D designs API inside DirectX) is broadly utilized as a part of the improvement of computer games for Microsoft Windows and the Xbox line of consoles. Direct3D is additionally utilized by other programming applications for perception and illustrations undertakings, for example, CAD/CAM designing. As Direct3D is the most broadly plugged part of DirectX, it is regular to see the names "DirectX" and "Direct3D" utilized conversely.

Chapter 3: Project planning and monitoring

Chapter 3: Project planning and monitoring

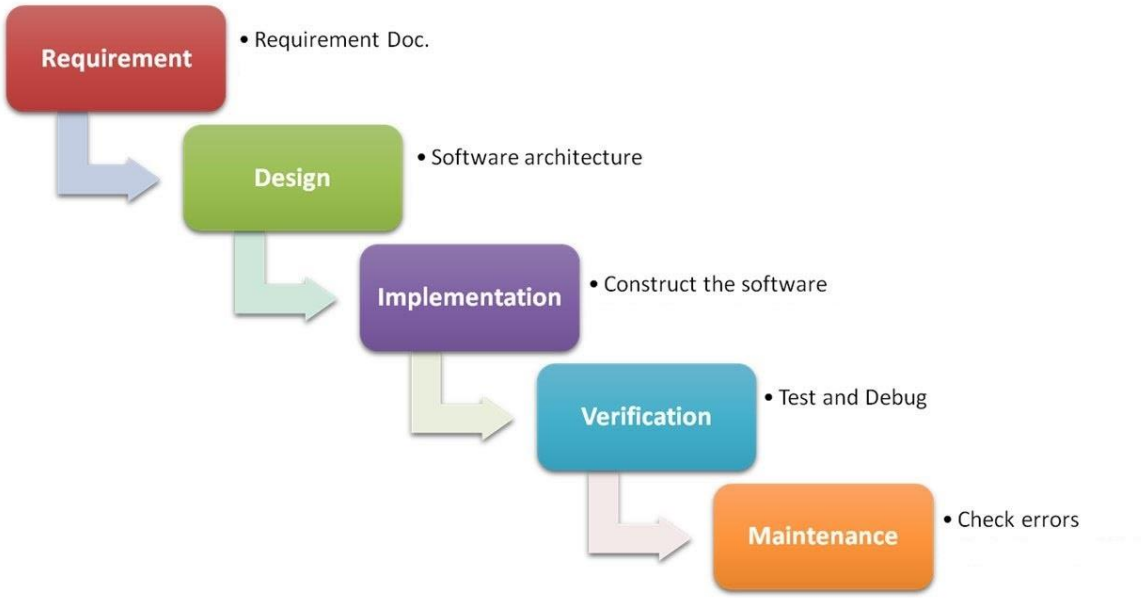


Fig. 8. Software development life cycle.

3.1: Functional Requirements

- The program will display a given 3D Model in free space, giving the user the ability to manipulate it in 3D Space.
- The user should be able to load/display any number of objects.
- The user can then select an object, and perform one or more subdivision operation on this object, making it smoother, smaller and increasing the number of vertex points in that object.
- The user can move these objects in free space
- On object collision, the 2 objects will be blended together into 1 smoothed object.

3.2: Non-Functional Requirements

- **Performance**
  - Response time should be less than 3 seconds.
  - Loading objects should be less than 5 seconds.
- **Reliability**
  - The ability to load any 3D model.
  - No internet connection required.
  - Real time results.
- **Security**
  - Security options have not been added to the system, as it is not needed.

3.3: Project Requirements

- **Hardware Requirements**
  - Computer with high processing power, and at least 4 gigabytes of ram.
  - Screen with high brightness.
- **Software Requirements**
  - .Net 4.0.
  - Microsoft visual studio 2015 (c#).
  - OpenGL.
  - Microsoft Windows 10

3.4: Class Diagram

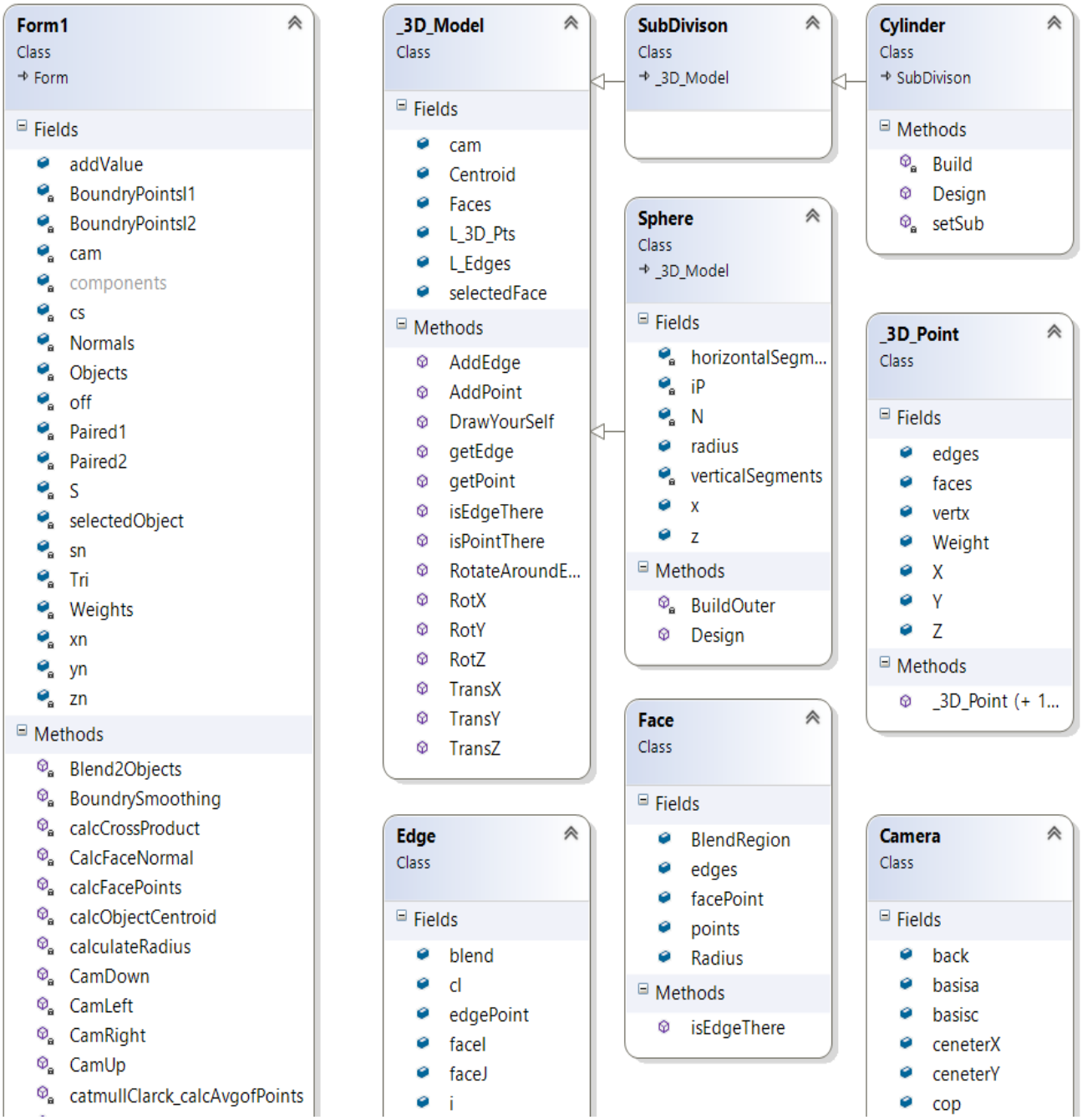


Fig. 9. Class Diagram (1)

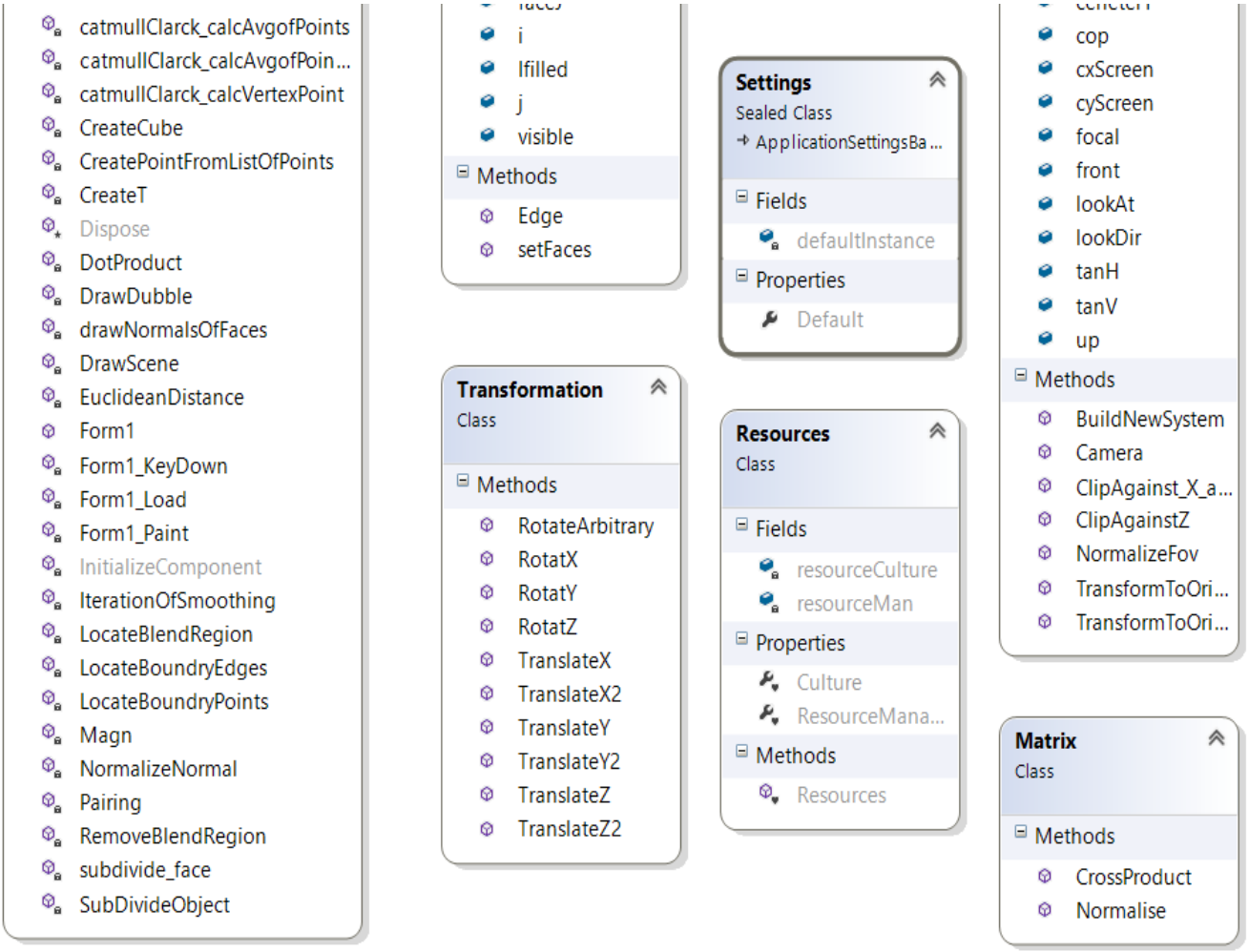


Fig. 10. Class Diagram (2)

3.5: Use Case Diagram

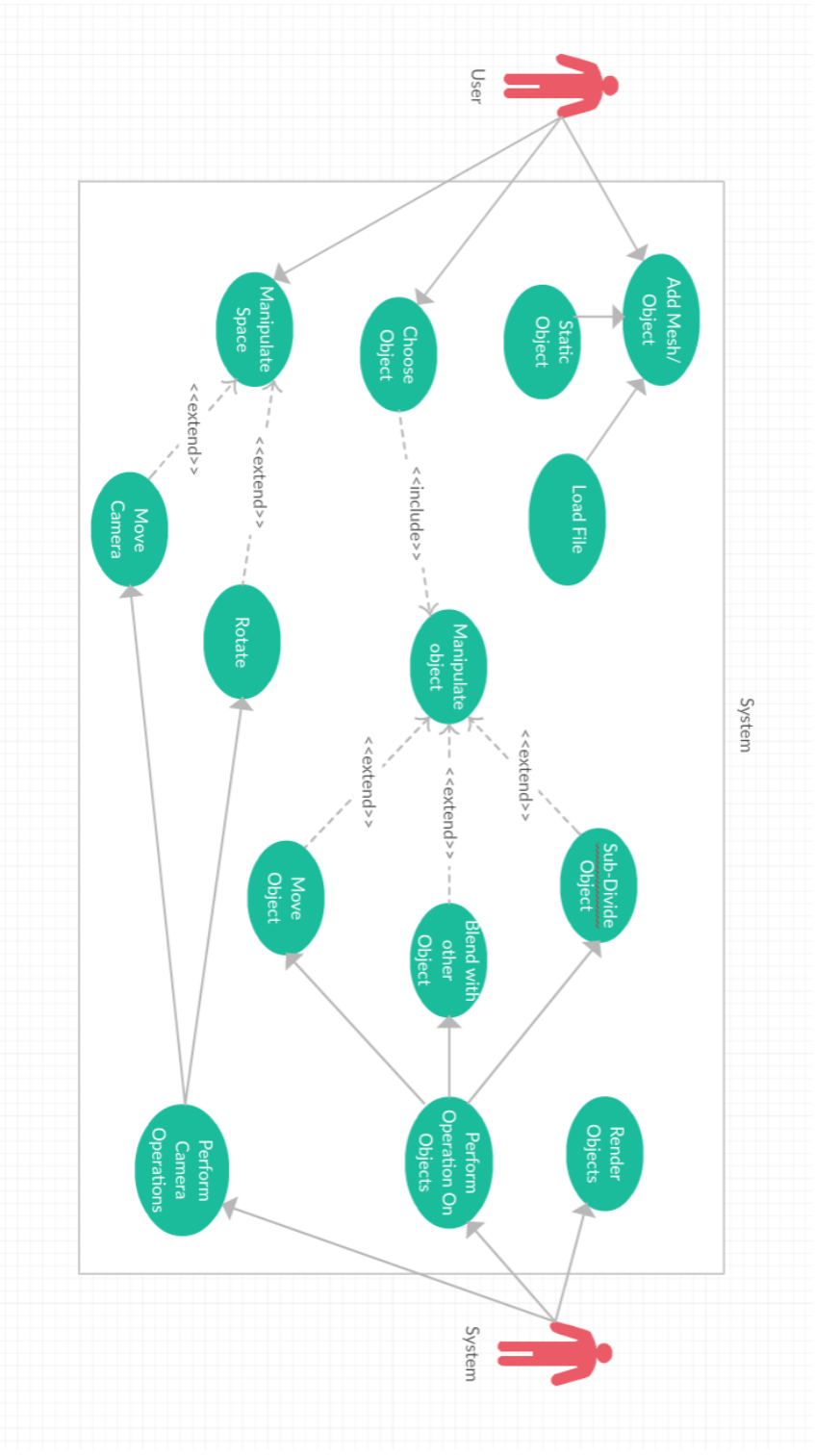


Fig. 11. (UML) Use Case Diagram

The user should be able to add an object/mesh either static or by loading an object file into the system. The user also have full control of the scene, meaning he can move the camera freely in space as well as zoom in and out, rotate the camera around and basically move in free space. The user than can choose one of the objects in the state space to perform a certain task or operation on that object, operations such as subdividing this particular object, moving the object, blending it with another object if they are close enough. They system should automatically respond to any user action choice by performing what's required.

Chapter 4: Implementation

Chapter 4: Implementation

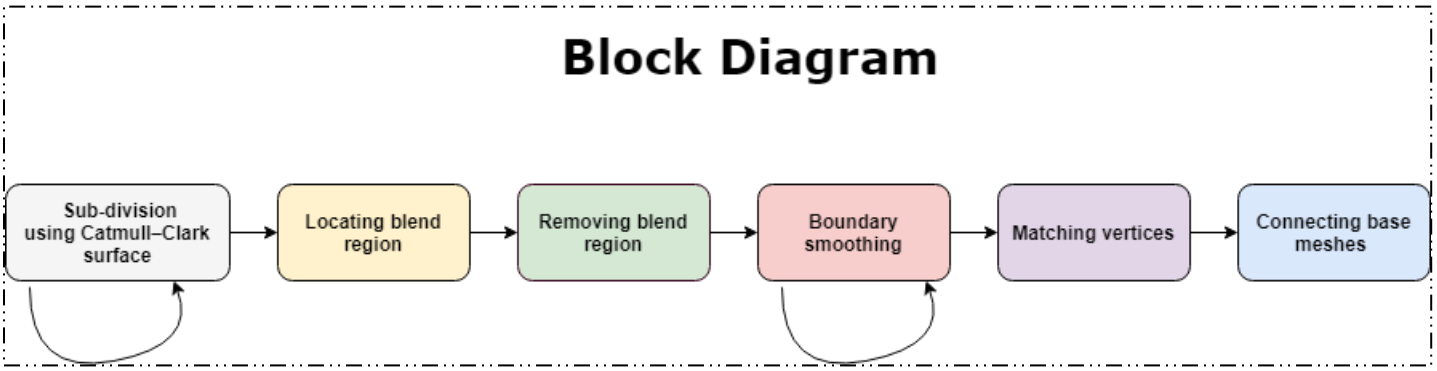


Fig. 12. System Block Diagram.

Method Overview

There are six stages of the proposed smooth blending technique for subdivision surfaces. They are the recursive process of sub-dividing the meshes using Catmull-Clark scheme, locating the blend region and removing it, the mesh refinement and boundary smoothing, the creation of vertex correspondence through matching vertices , and connecting the base meshes. In the first stage, a selected mesh will be sub-divided into a finer mesh of control-points that contains more polygon faces. This mesh can then be passed again through the same process. In the second stage, a blend region between two meshes is located with the aid of detecting intersecting faces at a user defined subdivision level for each of the intersecting meshes.

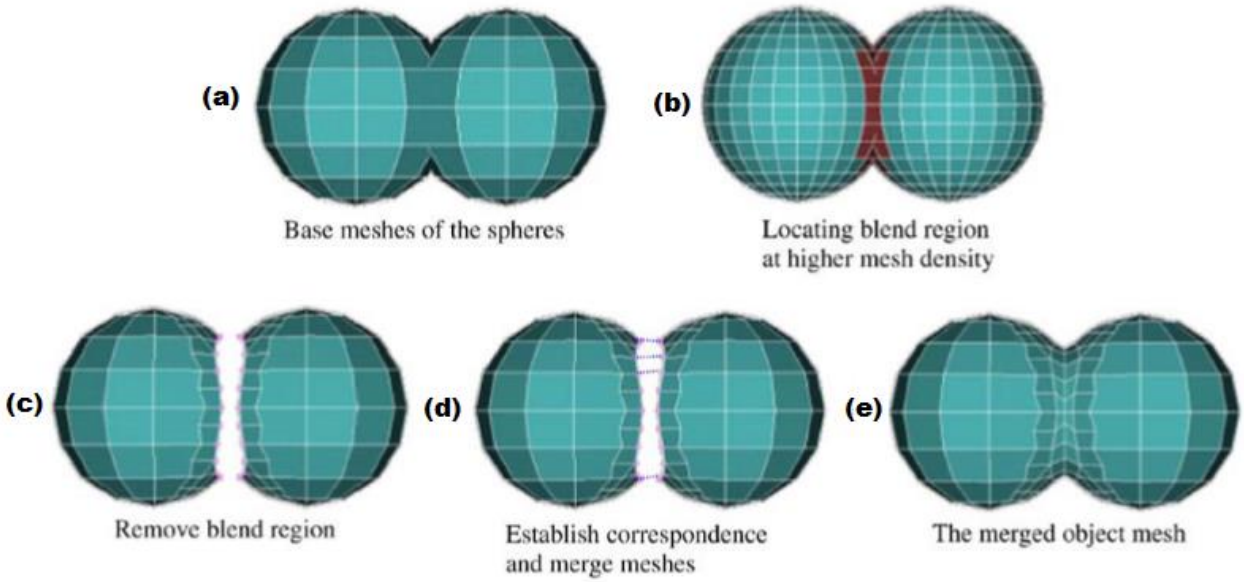


Fig.13. Overview of the proposed blending technique. [5]

Sphere-trees are used for approximating the objects and locating the intersection region. This will eliminate the requirement of calculating the intersecting curve, and thus eliminates the feasible numerical mistakes when the two objects touch. Fig. 13 (b) shows the blend region among two intersecting spheres. In the third stage, Polygons lying in the blend region are discarded as shown in Fig. 13 (c). In the fourth stage, the boundary curve where the blend region ends on both meshes is located, then the boundary points are smoothed out for a smooth blending effect using face normal combined with the original vertices positions. The fifth stage is the creation of vertex correspondence through matching vertices, preparing for stage six where the vertices will be connected through the blend curve.

4.1: Sub-division using Catmull–Clark surface

There are three basic steps to subdividing a mesh of control-points:

- Add a new point to every face, and a new point to every edge. Called the face-point and the edge point respectively.
- Reallocate the controlling point to another position, referred to as the vertex-point.
- Connect the newly inserted points and the reallocated vertex point. Adding new faces to the mesh.

Stage Input: A mesh of control points

Stage Output: A finer mesh of control-points that contains more polygon faces. This mesh can then be passed again through the same process.

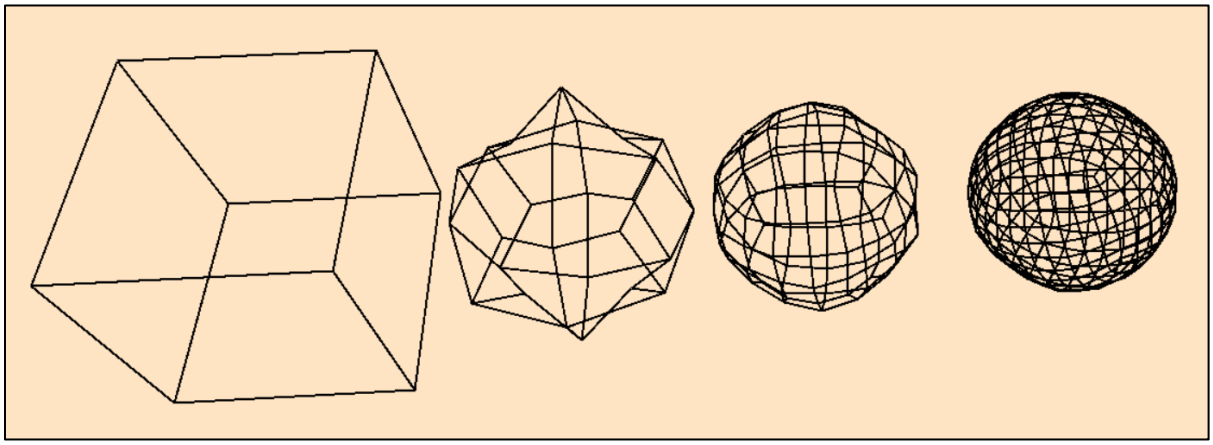


Fig. 14. Three iterations of sub-division using Catmull–Clark surface.

As shown in Fig. 14, the process can be done multiple times to produce finer meshes.

4.1.1: Step One

Inserting a face point to each face, and a edge face to each edge. The data structure for each face and edge will be updated after calculating these points. Starting with the face point that is the mean of all the controlling points of a face, usually four points in Catmull-Clark sub-division scheme.

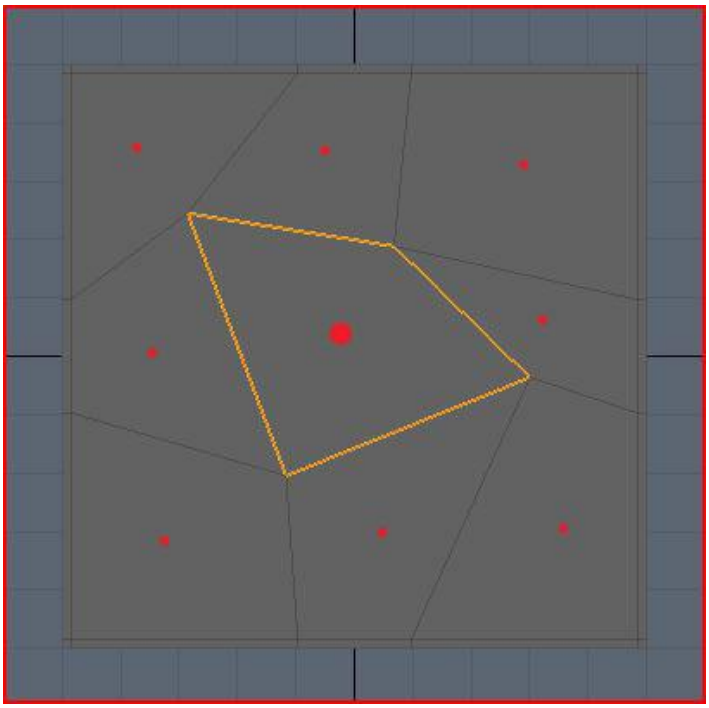


Fig. 15. Face points example. [1]

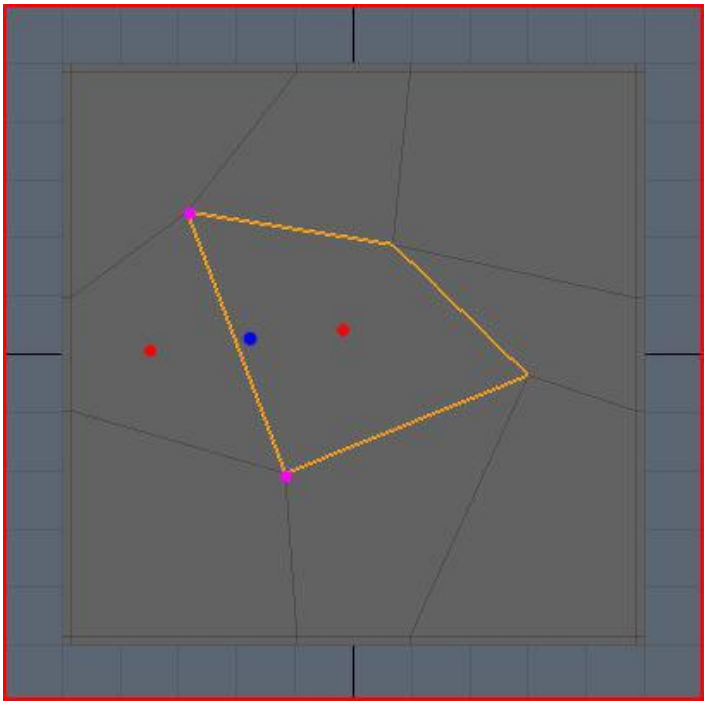


Fig. 16. Edge Point example. [1]

As shown in Fig. 15, the red dots are simply where the face points will be approximately, the big red dot is the face point of the yellow highlighted face. Each face point is the sum of all the controlling points of a face divided by the number of the controlling points.

Similarly for edge points, because we're not using weighted edges or boundary edges, they are also the average of the two points creating the edge and the two face-points of the neighboring faces, which makes it affected by four points total, as shown in Fig. 16, the blue dot is where the edge point will be located approximately. The pink dots are the two points on either side of the edge, the red dots are the face points of the neighboring faces. Keep in mind that edge points do not always lie on the original edge.

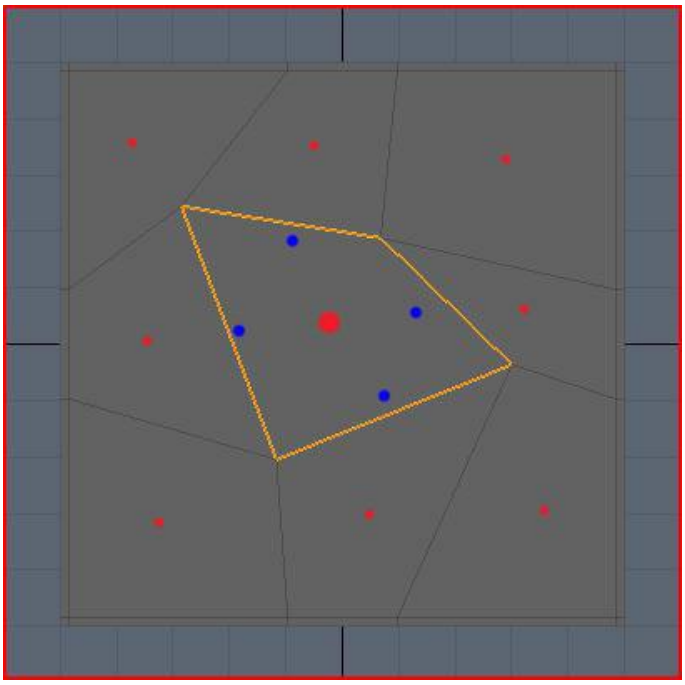


Fig. 17. Edge Points. [1]

As shown in Fig. 17, the blue dots are approximately where the edge points will go only for the yellow highlighted face.

4.1.2: Step Two

Step two is the reallocation of the vertex-points, and why their location changed from the original poison to the new one, but in this step weighted average must be used to determine the new location of the vertex-point. Weighted average is basically a normal average except that some of the values will have greater effect on the final result that other values.

Definition: valence

The valance of a point is the number of edges connected to that particular point. In the shown example mesh, you can clearly see that the valance is four for all the control points.

The new vertex-point location will be:

$$(Q/n) + (2R/n) + (S(n-3)/n)$$

Where Q is the average of the neighboring face points, R is the average of the surrounding edge midpoints, S is the original control point, and n is the point valence.

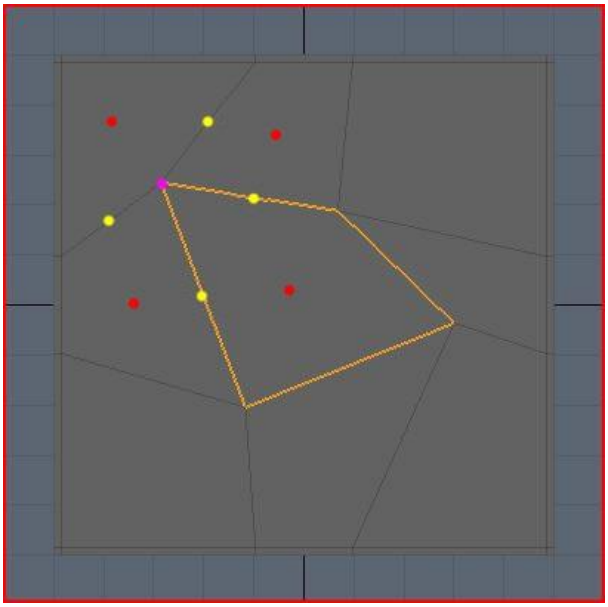


Fig. 18. Affecting points in locating the new vertex-point. [1]

As shown in Fig. 18, the Red dots represent (Q) average of the neighboring face points, the yellow dots represent (R) average of the surrounding edge midpoints, the pink dot represent (S) the original control point.

After applying a valence of four, the final equation will be:

$$(Q/4) + (R/2) + (S/4)$$

Which means that quarter of the weight of the new vertex point come from Q, Half come from R, and the last quarter from S.

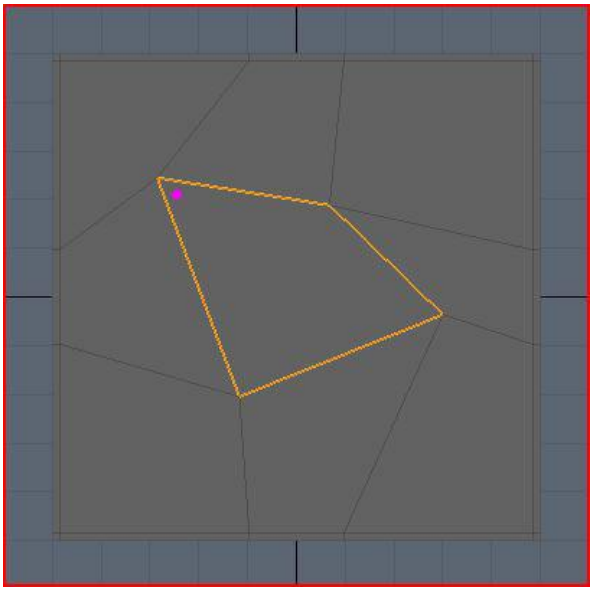


Fig. 19. New vertex-point location. [1]

In Fig. 19, the pink dot is approximately where the new vertex point will be. The control point is pulled down to the right, because all the surrounding points were averaged and weighted according to the equation to pull the vertex around, which also mean that if a vertex is next to large face and three small ones, the large face is going to pull the vertex to it more than the small ones do.

4.1.3: Step Three

In this step, all the newly inserted face points and edge points along with the reallocated vertex points are to be connected together to form four new faces

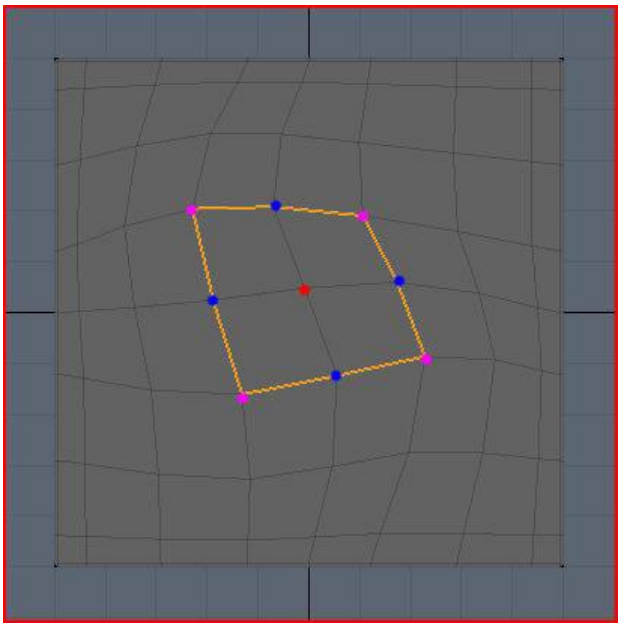


Fig. 20. Four new resulting faces. [1]

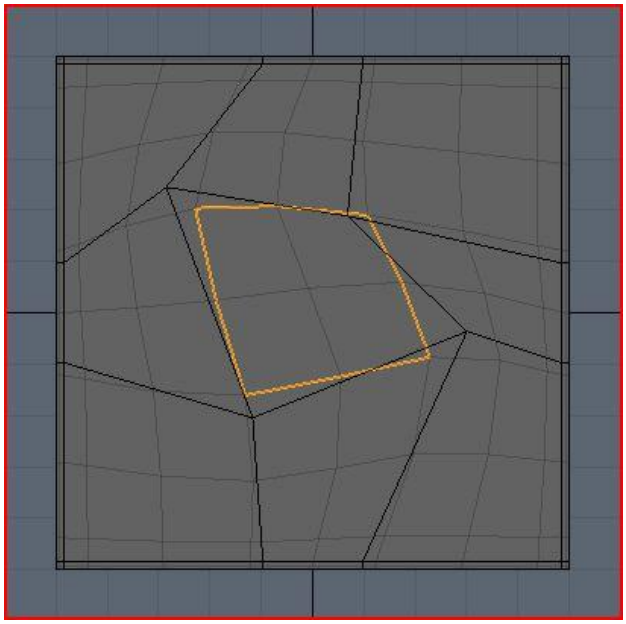
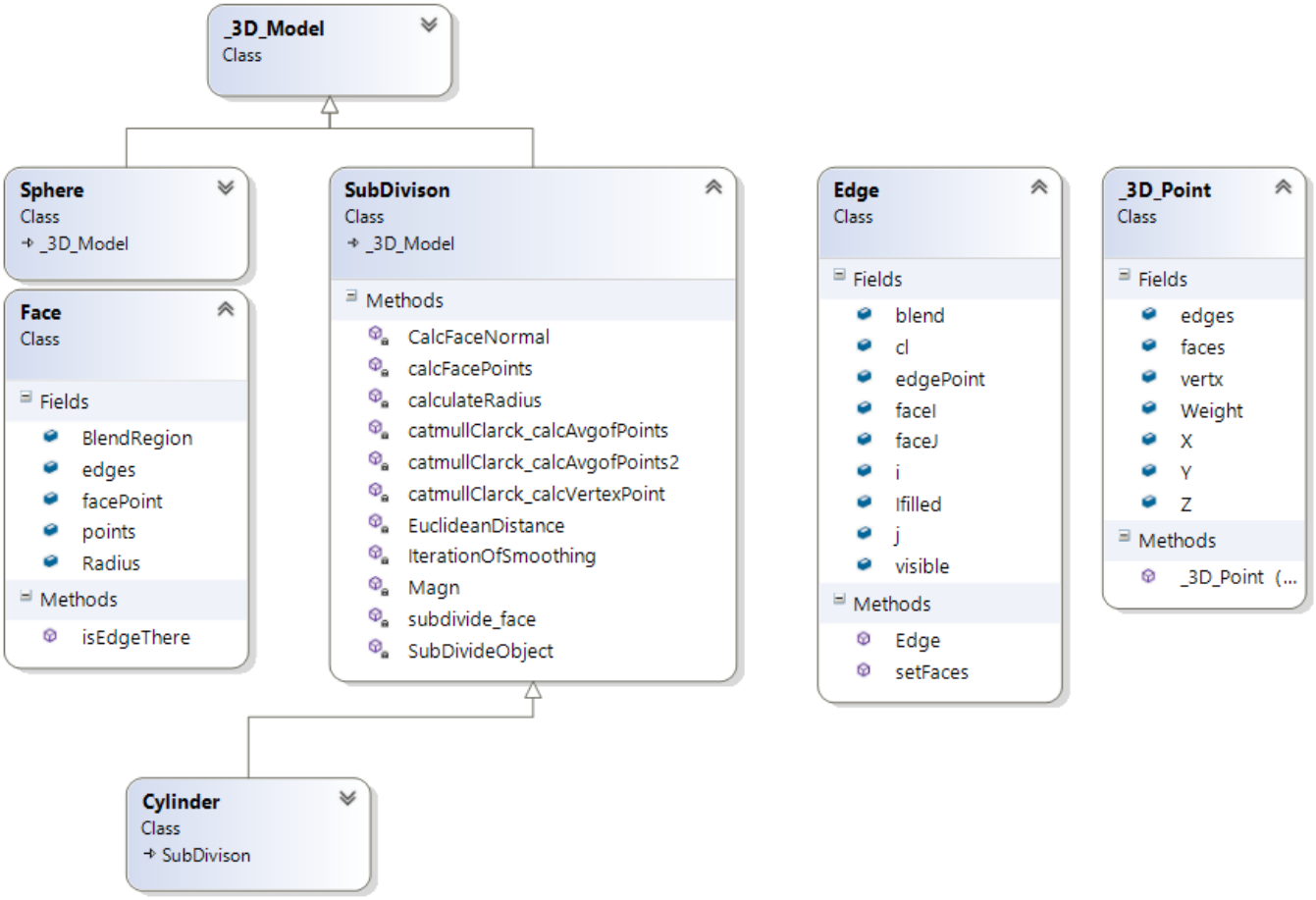


Fig. 21. Overview of before and after. [1]

In Fig. 20 only for the yellow highlighted face, the new face point is in red, the new edge-points are in blue, and the reallocated vertex-points are in pink.

Fig. 21 is an overview of how the original face looked like, and how the 4 resulting quad faces will look like. As mentioned earlier in step two, you can see that the top left controlling point is pulled down to the right. Same process happens for all other faces, resulting in four new quads added to the mesh.

4.1.4: Class Diagram



In the subdivision class we can see all the methods/functions being used to subdivide a certain 3D_Model, including calculating average of points, calculating face points, calculating vertex points and edge points, as well as performing the iteration of subdividing a passed object. Also a list of edges, points are needed to perform the mathematical equations in the subdividing scheme, as well as update these lists with the new inserted vertices, edged and faces.

4.2: Locating the blend region

In this stage, the blend region where the two subdivided meshes may intersect must be located. This is to locate the intersecting faces between the two base meshes.

Stage Input: two intersected meshes

Stage Output: Locating the intersected polygons of the two intersected meshes.

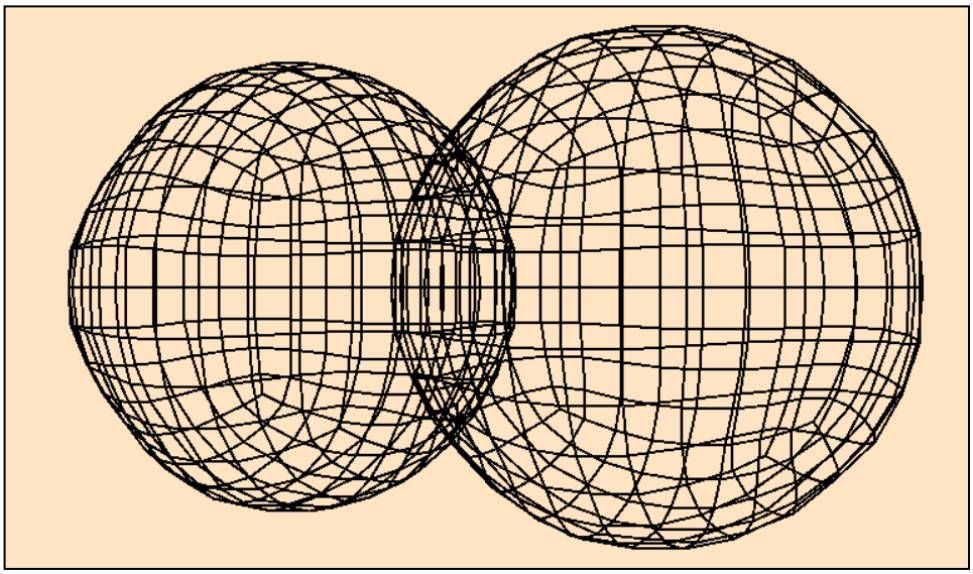


Fig. 22. Stage (2) Input

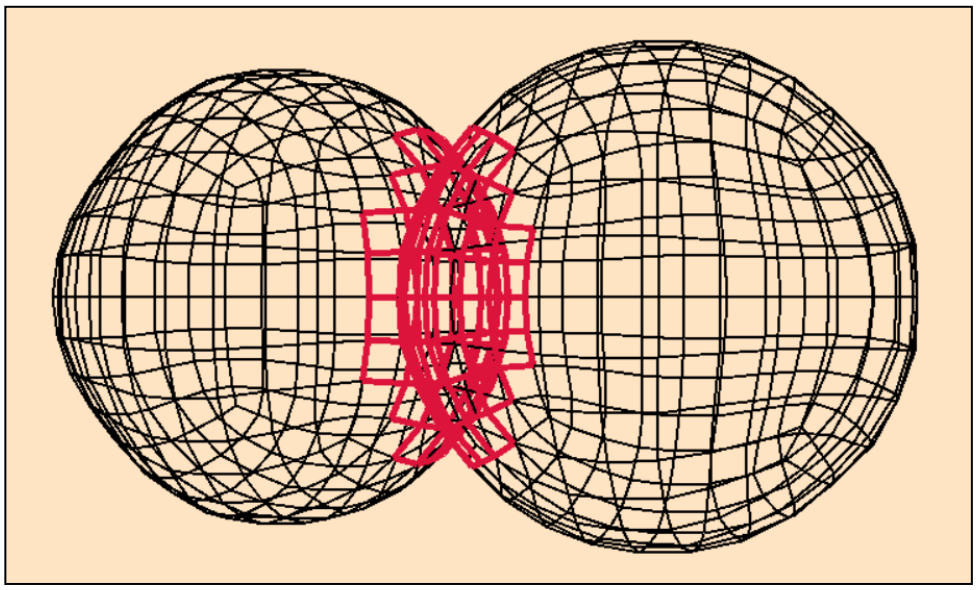


Fig. 23. Stage (2) Output

4.2.1: Blend region

The base meshes P, Q of the two subdivision surfaces are Sp, Sq respectively. Assuming polygons of the base meshes are either triangles or quadrilaterals, as well as assuming edge that are of more or less the same lengths, then there is no sharp angle between the edges. Denote P^l , Q^m as the polygon meshes of Sp, Sq at the lth and mth subdivision stages respectively. Since Catmull-Clark subdivision scheme is used in this method, a triangle is transformed into a set of quadrilaterals in a single subdivision iteration, and then the shape of a quadrilateral is no longer affected in a subdivision technique as it stays the same.

The form of the polygons of P^l , Q^m are basically the same as those in P^1 and Q^1. Assuming the polygon sizes of the two polygon meshes P^l , Q^m are close in area, then

$\forall p \in P^l$ and $q \in Q^m$

$|A(p) - A(q)| < \varepsilon$, where A(x) is the area of a polygon x, and ε a specified threshold, the blend region of Sp and Sq is the region where P and Q are intersected.

Definition: Blend region

The Blend region is the l by m level blend region (B^{lm}_{pq}) of two surfaces Sp, Sq with base meshes P, Q respectively, the set of polygons Y such that

$Y \in P^l$ and $Y \cap Q^m \neq \varphi$, or $Y \in Q^m$ and $Y \cap P^l \neq \varphi$.

So locating the blend region means to determine the subdivision level l an m as well as detecting all intersecting polygons in P^l , Q^m. Since Catmull-Clark surface scheme is used for P and Q, then moth base meshes of , that is, the base meshes of P^0 and Q^0 are composed of quadrilaterals, a subdivision of a polygon $p^0 \in P^0$ will give another four quadrilaterals. If the sizes of all polygons P^1 in the lth subdivision level are the same, then:

$$A(p^{l+1}) = \frac{1}{4}A(p^l),$$

or,

$$A(p^l) = \frac{1}{4^l}A(p^0). \quad (1)$$

Hence, the condition $|A(p^l) - A(p^m)| \leq \varepsilon$ is satisfied if

$$\left|\frac{1}{4^l}A(p^0) - \frac{1}{4^m}A(q^0)\right| \leq \varepsilon. \quad (2)$$

Also assuming that P^0 and Q^0 are more or less squared in terms of shape. Which means there are no sharp angels between edges. Denote L_p and L_q as the length of the edges for P^0 and Q^0 respectively. Then, area of $P^0 = L_p^2$ and area of $Q^0 = L_q^2$ and equation (2) can be re-written as follows:

$$\left| \frac{L_p^2}{4^l} - \frac{L_q^2}{4^m} \right| \leq \varepsilon. \qquad (3)$$

Since l and m are integers, the minimum values of l and m give the minimum subdivision levels for locating the blend region. Because P^l , Q^m are compatible in size, anymore subdivision of meshes P^l , Q^m gives similar polygons in terms of size as well.

4.2.2: Sphere Trees for collision detection

Since an exact intersection of Sp and Sq is now not necessary, a simple collision detection method was adopted to locate the blend region. A sphere-tree is build for each of Sp and Sq. A sphere of the sphere-tree encloses a certain polygon of the surface, denote $h^l_{p,i}$ as a sphere at the lth level of the sphere-tree of S_p enclosing a polygon p_i^l . Similarly, $h_q^{m,\,j}$ denotes a sphere at the mth level of the sphere-tree of S_q enclosing a polygon q^m_j . If $h^l_{p,i} \cap h_q^{m,\,j} \neq \varphi$, then p_i^l and q^m_j may possibly intersect. The polygons p_i^l and q^m_j can then be discarded. Since a precise intersection is not required, the blend region can be located for some objects that do not intersect but are close enough.

In some situations the polygons p_i^l , q^m_j are parallel, which happens when S_p and S_q meet in the region where the spheres $h^l_{p,i}$ and $h_q^{m,\,j}$ intersect, removing p_i^l and q^m_j may remove the parallel region of the object entirely. This may result in a blend surface that deviates significantly from the original surface as shown in Fig. 24(c). This undesirable effect is eliminated by considering the neighboring polygons of p_i^l and q^m_j . If p_i^r is a polygon adjacent to p_i^l , and the angle between p_i^r and p_i^l is less than $180^\circ - \delta$, where δ defined threshold, then p_i^r is not removed. This retains the polygons on the boundary of the parallel region, and thus confines the shape of the blend surface to more closely resemble the original surfaces as shown in Fig. 24(e).

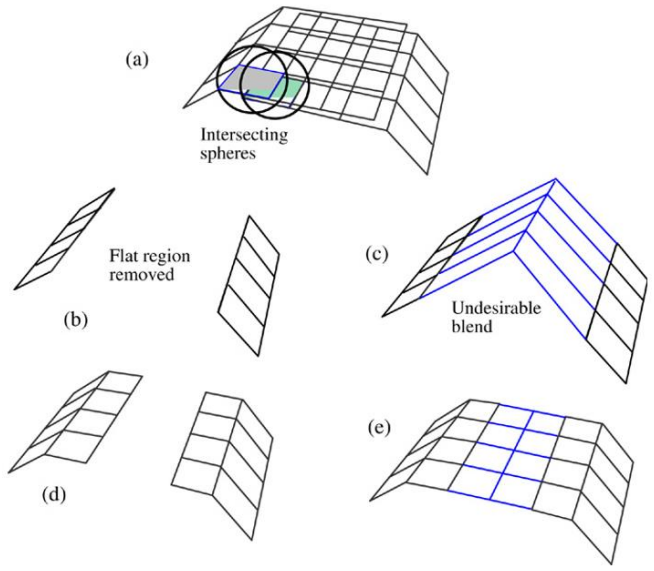
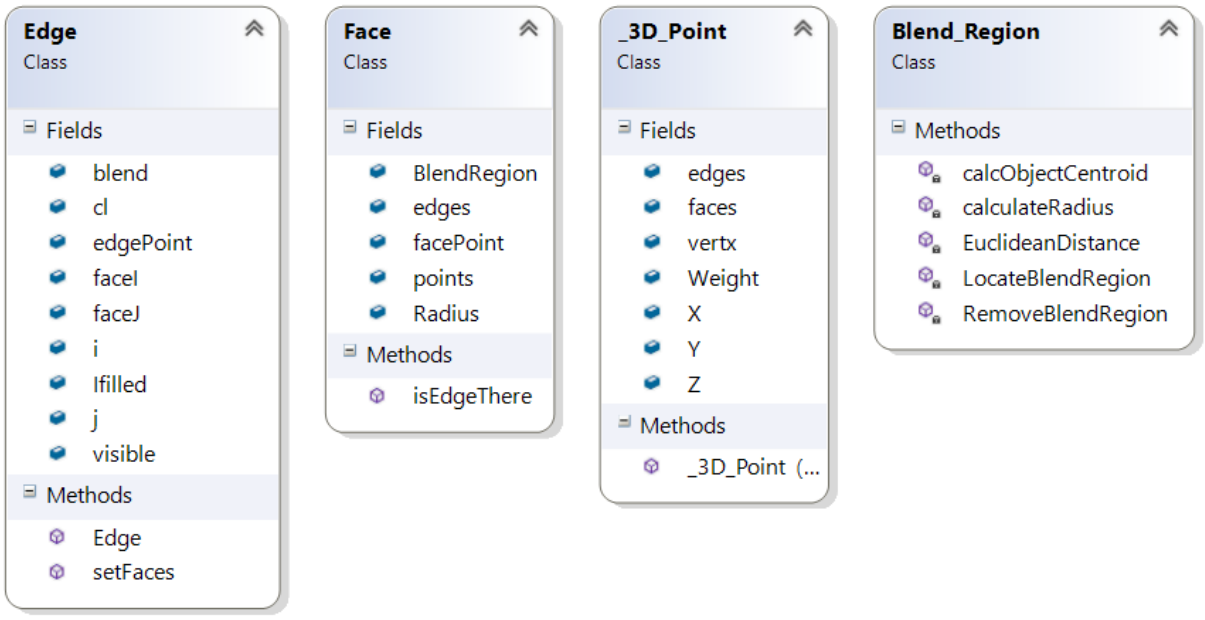


Fig. 24. Undesirable blend. [5]

4.2.3: Class Diagram



Locating the blend region required several methods including EuclideanDistance, locateBlendRegion that uses sphere trees to locate the blend region.

4.3: Removing the blend region

In this stage, and after the collection of intersected polygons between the two meshes have been determined, the blend region will be discarded.

Stage Input: List of intersected polygons between two meshes.

Stage Output: Discarding these polygons, determining a list of boundary points and edges.

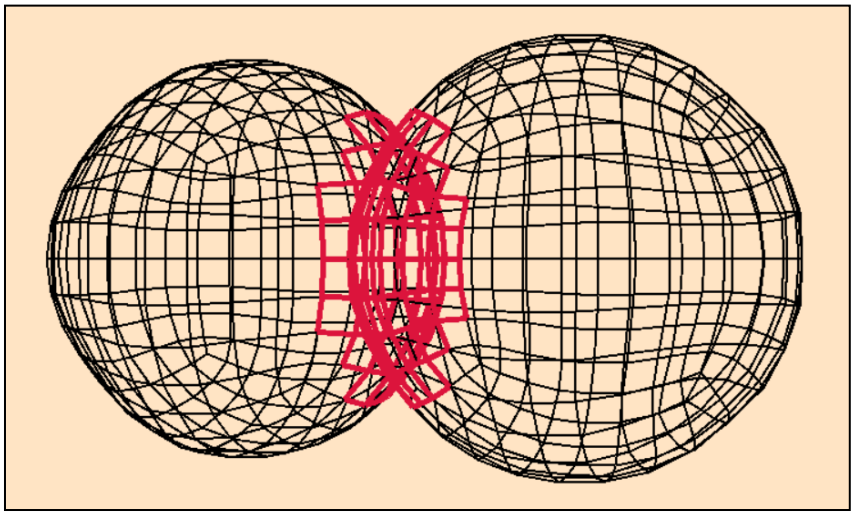


Fig. 25. Stage (3) Input

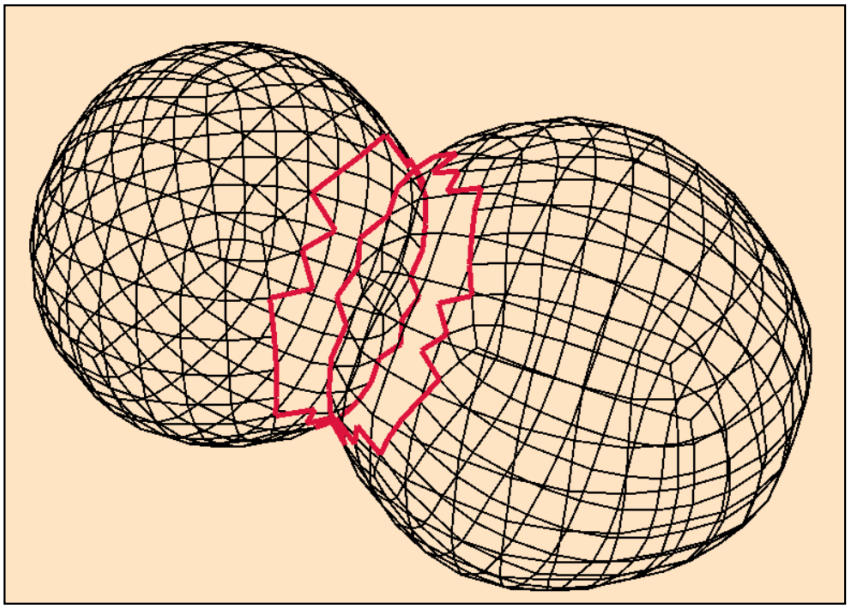


Fig. 26. Stage (3) Output

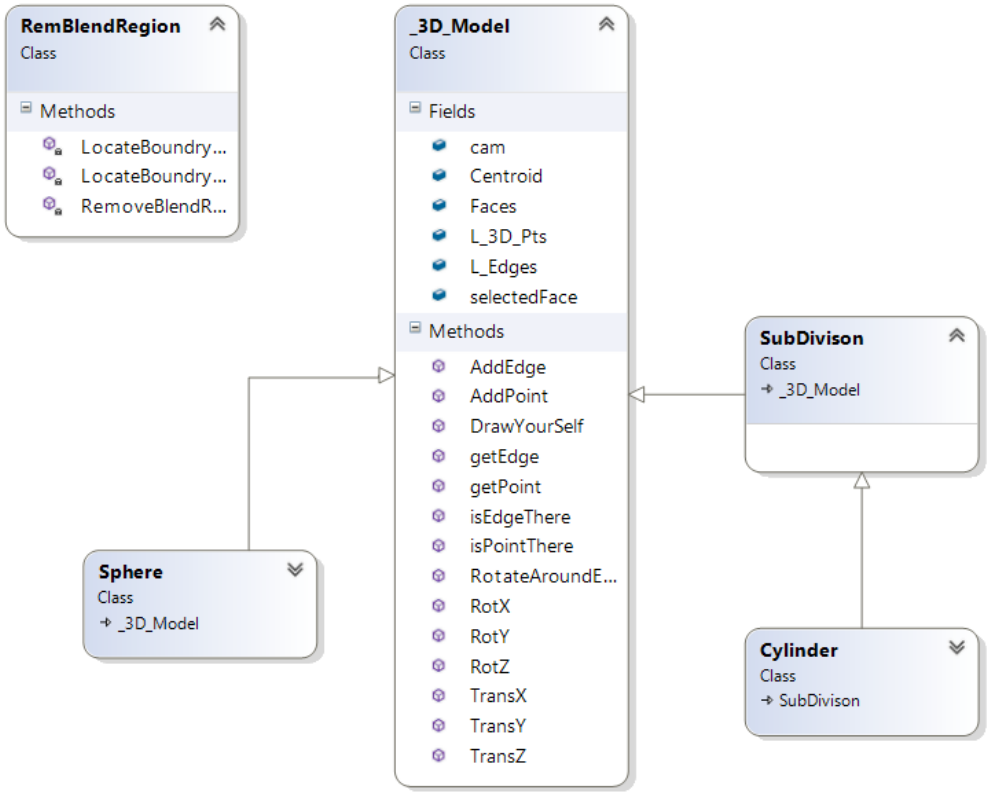
4.3.1: Discarding polygons

Discarding the list of intersected polygons requires multiple updates in the data structures of each of the two intersected meshes.

- Removing the set of polygons/faces for each of the two objects.
- Removing some edges that do not belong to any polygon/face.
- Removing some points that do not belong to any edge.
- Updating existing polygons of the removed points.
- Updating existing polygons of the removed edges.
- Updating existing edges of the removed polygons.
- Updating existing points of the removed polygons.
- Updating indexes for all lists.

As a result, after discarding all undesired polygons of the blend region, a boundary curve appears on both objects where the blend region used to be. Boundary points and Boundary edges lying on the boundary curve must be located in preparation for the boundary smoothing stage.

4.3.2: Class Diagram



4.4: Boundary smoothing

After discarding the blend region polygons, the resulting boundary curves may have sharp vertices, these sharp corners located on the boundary of the blend region will eventually become sharp corners on the boundary of the blend area, which will eventually lead to a surface with waves. In this stage, the process of smoothing the boundary points is to eliminate these sharp corners and it is done recursively.

Stage Input: List of Boundary points and Boundary edges.

Stage Output: Smoothed out Boundaries along both meshes.

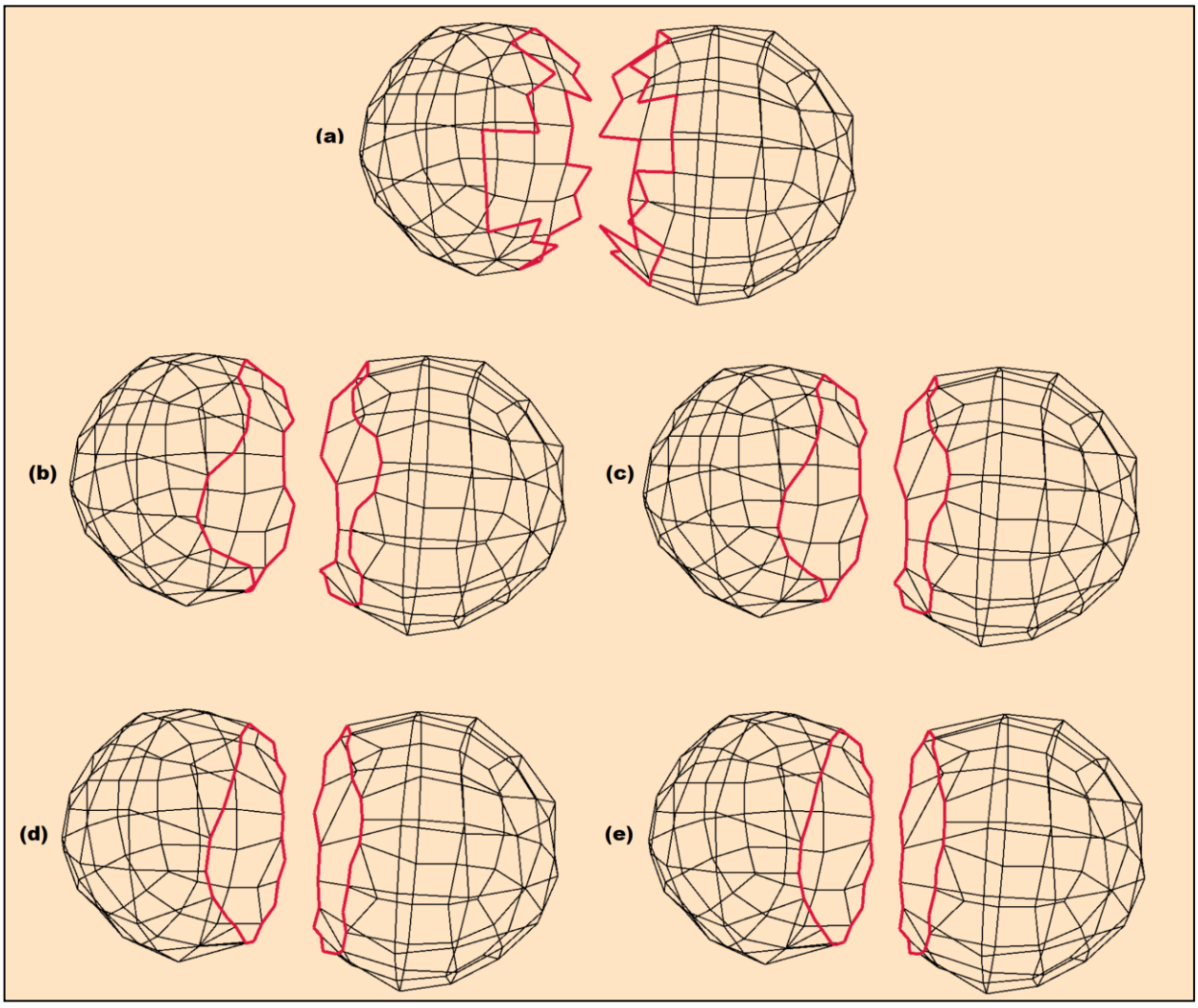


Fig. 27. Boundary Smoothing iterations

Fig. 27(a) shows the input meshes with no boundary smoothing done yet. Fig. 27(b) shows the first iteration of boundary smoothing, Fig. 27(c) shows two iterations of smoothing, Fig. 27(d) shows three iteration of smoothing and Fig. 27(e) shows four iterations of smoothing.

4.4.1: Iterative process

Fig. 28 shows the sharp vertices resulting from discarding the blend region polygons. A boundary smoothing process is expressed as:

$$\mathbf{p}_i^r = \frac{\mathbf{p}_{i-1}^{r-1}}{4} + \frac{\mathbf{p}_i^{r-1}}{2} + \frac{\mathbf{p}_{i+1}^{r-1}}{4} \tag{4}$$

where $\mathbf{p}_{i-1}^{r-1}, \mathbf{p}_i^{r-1}, \mathbf{p}_{i+1}^{r-1}$ are three consecutive boundary vertices at the $(r-1)$ iteration.

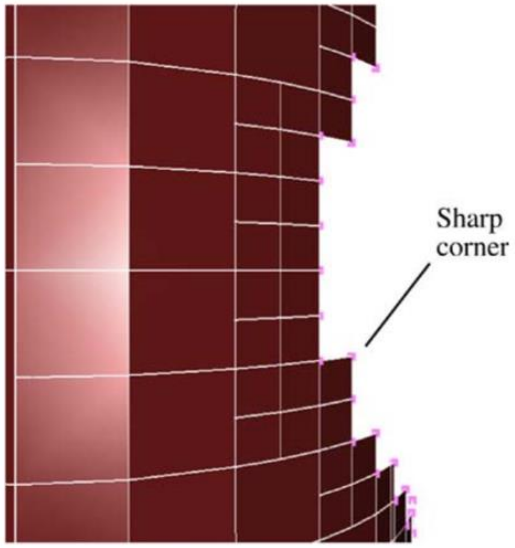


Fig. 28. Sharp Corners. [5]

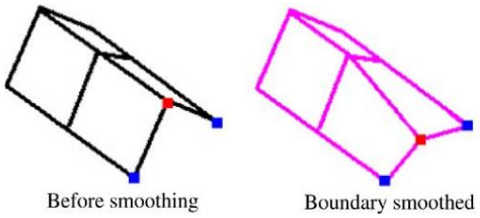


Fig. 29. Distortion example. [5]

The iteration from Eq. (4) may in some cases cause a distorted boundary as shown in Fig. 29. The solution to this is adding an additional constraint to make sure that the vertices will only be modified on the tangent plane of the surface. The equation is then written as follows:

$$\mathbf{p}_i'^r = \frac{\mathbf{p}_{i-1}^{r-1}}{4} + \frac{\mathbf{p}_i^{r-1}}{2} + \frac{\mathbf{p}_{i+1}^{r-1}}{4} \tag{5a}$$

$$\mathbf{d}_i^r = \mathbf{p}_i'^r - \mathbf{p}_i^{r-1} \tag{5b}$$

$$\mathbf{p}_i^r = \mathbf{p}_i'^r - (\mathbf{d}_i^r \bullet \mathbf{n})\mathbf{n} \tag{5c}$$

where $\mathbf{n}$ is the surface normal at $\mathbf{p}_i$.

The iteration in Eq.(5) can be done multiple times recursively to smooth the boundaries properly. Fig. 30(a) shows the intersecting objects, Fig. 30(b) shows the blending process done without boundary smoothing, and Fig. 30(c) shows the blending process done with boundary smoothing.

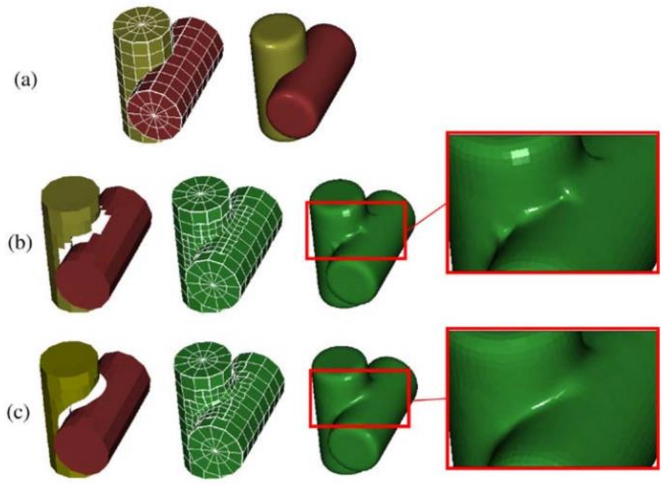
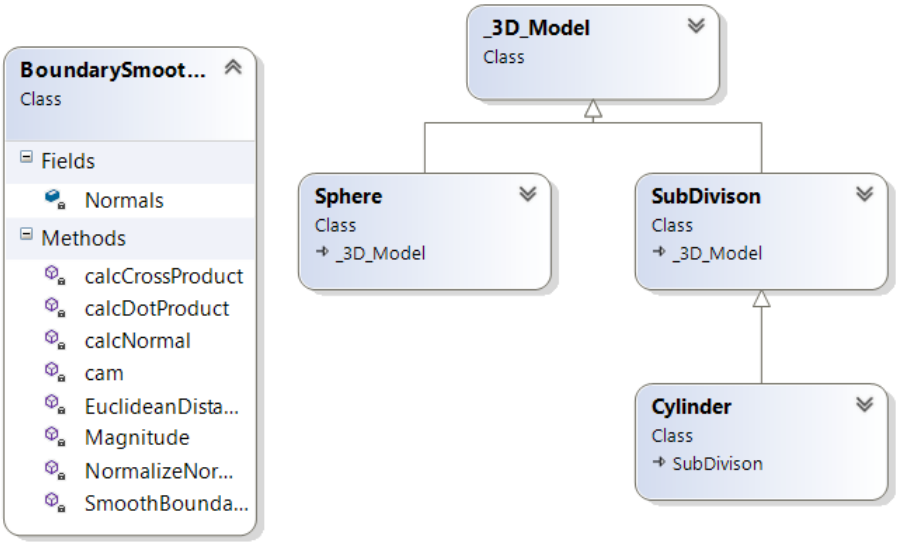


Fig. 30. Boundary Smoothing. [5]

4.4.2: Class Diagram



In the boundary smoothing class diagram, several method are essential for the process, including calculating the cross product, calculating the dot product, calculating the normal of a face, Euclidean distance and the magnitude, all are being used to do one iteration of boundary smoothing through the function SmoothBoundary.

4.5: Matching vertices

After smoothing the boundaries of the two subdivision surfaces, the simplest approach to connecting the base meshes is to connect corresponding points on the two boundaries instantly. However, extraordinary vertices might exist near the boundary curves. In this stage, the controlling points of the meshes along the boundary curves are to be paired together in preparation for connecting them to a new constructed blend curve lying inside the blend region as shown in Fig. 31.

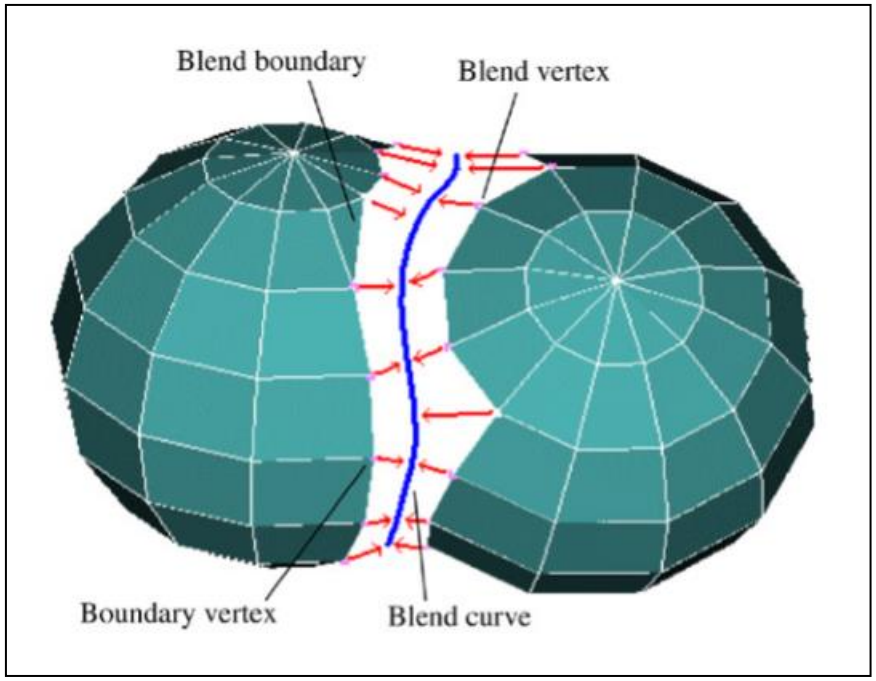


Fig. 31. Blend Curve. [5]

Stage Input: List of Boundary points and Boundary edges (After smoothing).

Stage Output: Vertex pairs.

Generally, the smoothness of a Catmull-Clark surface in a high curvature region is mostly dependent on the valence Vi of the polygon vertices Pi. Considering a valence of four, usually will give a smooth region around Pi whereas a valence that does not equal four may rise to unwanted ripples around Pi. Since the blend region is where the two different meshes intersected, high curvature would usually be found near the blend region. Therefore, to minimize the effects of the extraordinary points on the overall smoothness of the blending region, a blend curve C will be constructed and used for making the connections between the polygon meshes. With two intersecting surfaces with base meshes P, Q. While also assuming P^0 and Q^0 to be the

corresponding polygon meshes with the blend region discarded and the boundary curves Cp, Cq smoothed out. Where P^0 and Q^0 are connected lies the blend curve C, the valence of the vertices on the new constructed blend curve will be maintained at four. In addition, the blend curve's position is where the blend region used to be, extraordinary vertices with valence not equal to four would be eliminated and as a result reduce the possible distortion on the surface.

4.5.1: Pairing Vertices

In order to keep a valence of four amongst all vertices lying on the blend curve, each vertex of the blend curve C is connected to a vertex of Cp and a vertex of Cq. Which in return requires pairing vertices and establishing correspondence between vertices Cp and Cq. Since the blend region is located with subdivision levels at the base meshes, and the polygons are assumed similar to a rectangular shape more or less, the blend region's width would be relatively constant.

Denoting Pi, Qj, where i = 0, . . . , m-1

where j = 0, . . . , n-1 as the vertices on Cp, Cq respectively. The pairing process between vertices is established by matching Cp and Cq according to the following definitions:

Definition: Closest Vertex

Given Pi a vertex on Cp, and Qj a vertex on Cq, Qj is the Closest vertex to Pi if and only if

$$|\mathbf{p}_i - \mathbf{q}_j| < |\mathbf{p}_i - \mathbf{q}_k|, \ \forall k \neq j.$$

Definition: Vertex Pair

Given a vertex p on Cp, a vertex q on Cq is it's corresponding pair (q, p) if q is the Closest vertex to p, as well as p is the Closest vertex to Q.

Since there might be a difference in the mesh densities of the two surfaces resulting from using two different subdivision levels, the pairing process must be applied iteratively to match vertices on Cp and Cq. Fig. 32 shows the process with two iterations.

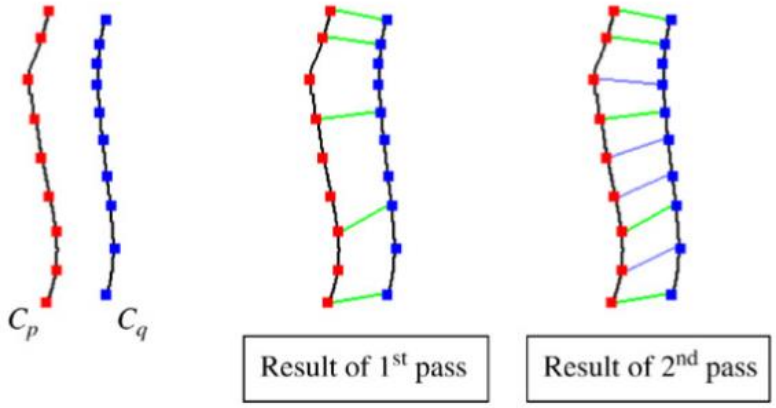
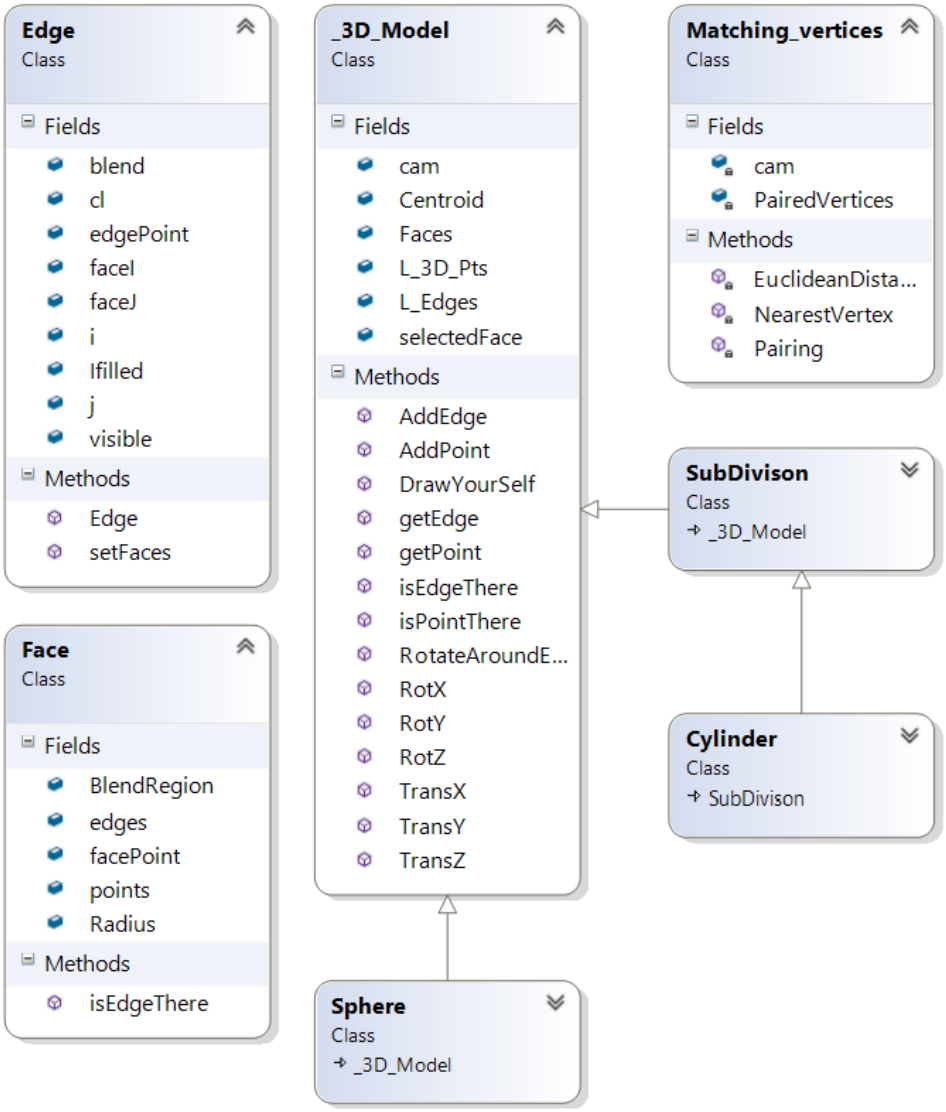


Fig. 32. Two iterations of Pairing. [5]

A corresponding pair for each vertex of Cp was located in the first iteration, a corresponding vertex for each unpaired vertex of Cq is located in the second iteration. This process must be repeated until no more pairs can be created. Since the determining factor in the number of vertices on Cp and Cq are the size of subdivided polygon, and the sizes of intersecting polygons p_i^l and p^m_j are more or less the same, and assuming the polygon shapes are similar as well, the difference in vertices number between Cp and Cq is not big. For the unmatched vertices, they become extraordinary lying on Cp and Cq, these extraordinary vertices are no restricted to lie in low curvature regions which in return reduces the undesired distortion.

4.5.2: Class Diagram



In the matching vertices class, the 2 most important methods are NearestVertex, which determines what is the nearest vertex out of a group to a certain vertex, and a Pairing function to check if both vertices are the nearest to each other. Keeping in mind that NearestVertex is using Euclidean distance to determine the distance.

4.6: Connecting base meshes

After successfully pairing the boundary points of both meshes, a blend curve must be established for connecting the base meshes together and generating the blended mesh. The blend curve will be located to keep the extraordinary vertices lying on a flat region of the surface and reduce distortion if possible.

Stage Input: Two lists of the paired vertices.

Stage Output: Complete blending of the two intersected meshes.

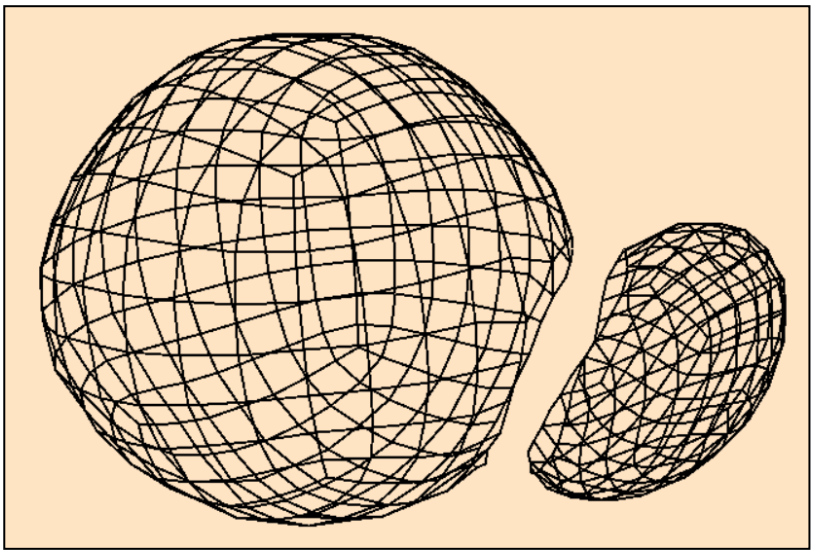


Fig. 33. Stage (6) input.

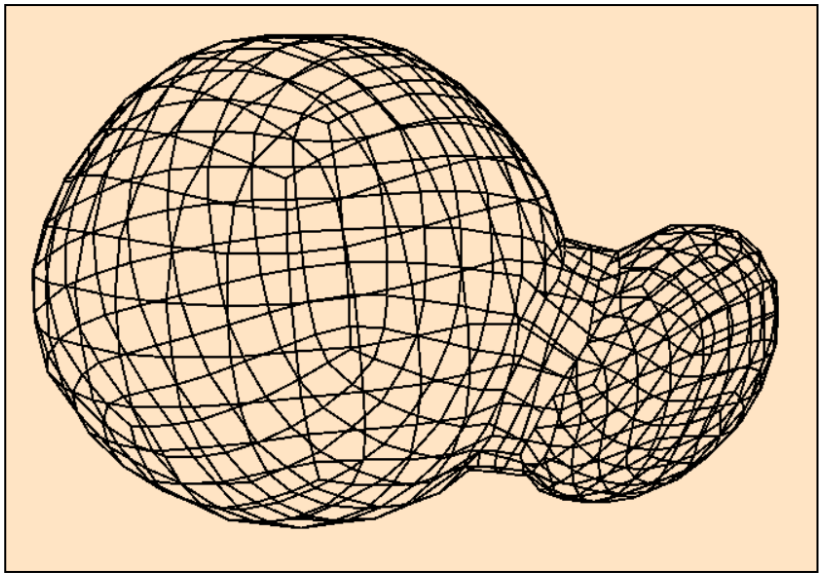


Fig. 34. Stage (6) output.

4.6.1: Creating the blend curve

As mentioned before in section 4.5.1, each vertex of the blend curve must be connected to a vertex of Cq, and a vertex of Cp, as well as connecting the curve points themselves together, which gives a valence of four amongst all vertices lying on the blend curve.

For each pair of vertices, a new point will be created and added to the blend curve, the new point locations determined according to the following equation:

$$(P_{i-1} W3) + (P_i W1) + (P_{i+1} W4) +$$
$$(P_{j-1} W5) + (P_j W2) + (P_{j+1} W6)$$

Where $\mathbf{P}_{i-1}, \mathbf{P}_i, \mathbf{P}_{i+1}$ are the three consecutive boundary vertices lying on the boundary curve of the first mesh, represented in red in Fig. 35. And $\mathbf{P}_{j-1}, \mathbf{P}_j, \mathbf{P}_{j+1}$ are the three consecutive boundary vertices lying on the boundary curve of the second mesh, represented in green as shown in Fig. 35. And W1, W2, W3, W4, W5 and W6 are predefined weights.

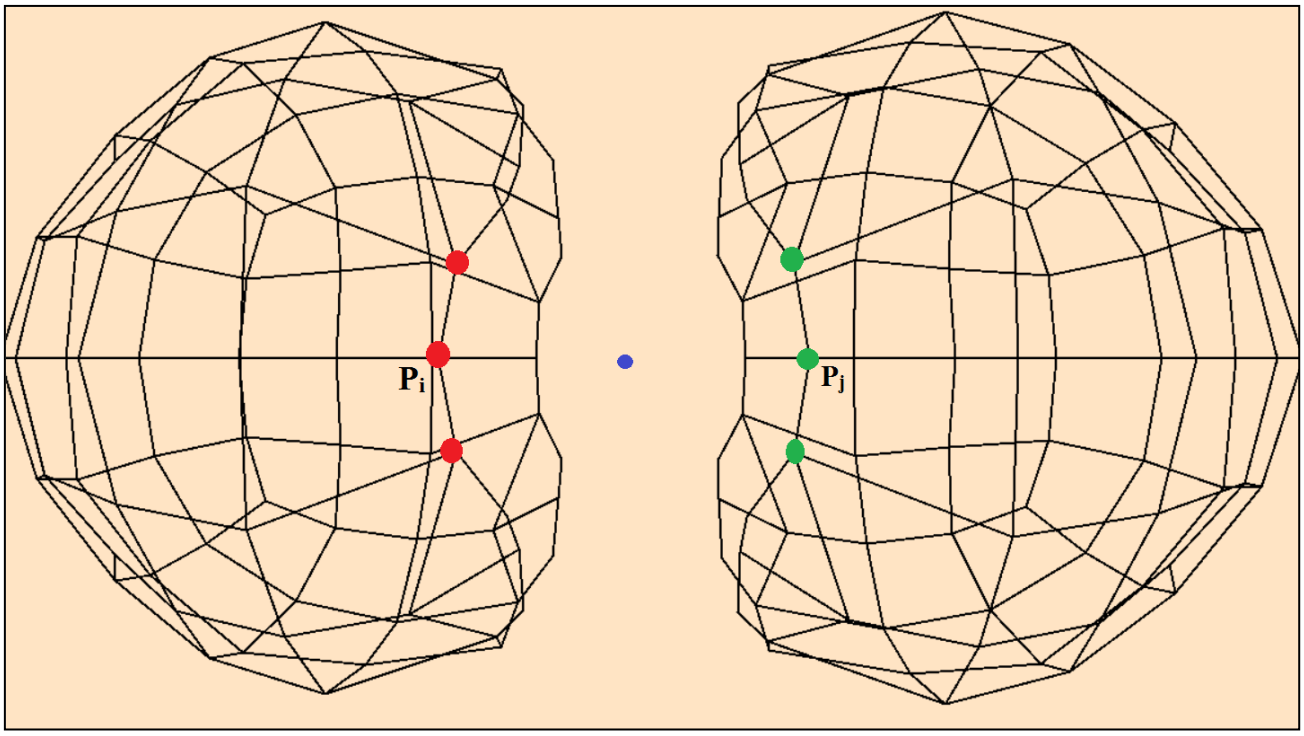


Fig. 35. Blending curve creation.

As shown in Fig.35. the red dots are $\mathbf{P}_{i-1}, \mathbf{P}_{i}, \mathbf{P}_{i+1}$, and the green dots are $\mathbf{P}_{j-1}, \mathbf{P}_{j}, \mathbf{P}_{j+1}$ are the blue dot is roughly where the new curve points will be located. This is done to all vertex pairs. Fig. 36 shows the topological rules for the weights distribution.

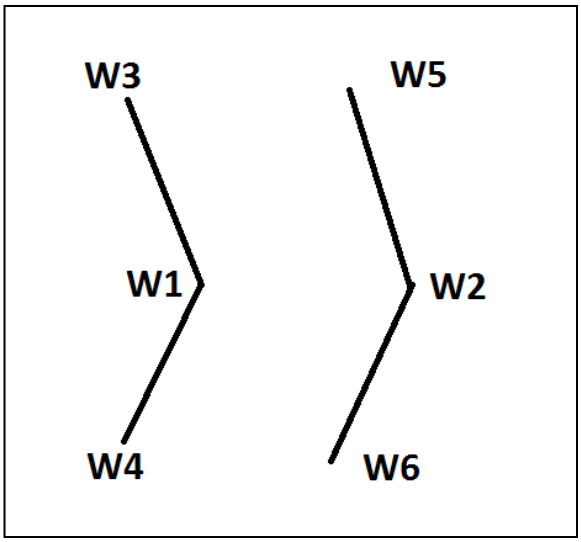


Fig. 36. Weights topological rules.

As for the weights, collections of different weights have been applied to the equation, each group of weights gives different results, because each group influences the value of the final point based on the six points we account for. These weights must sum up to a total of 1. Here are the proposed weights.

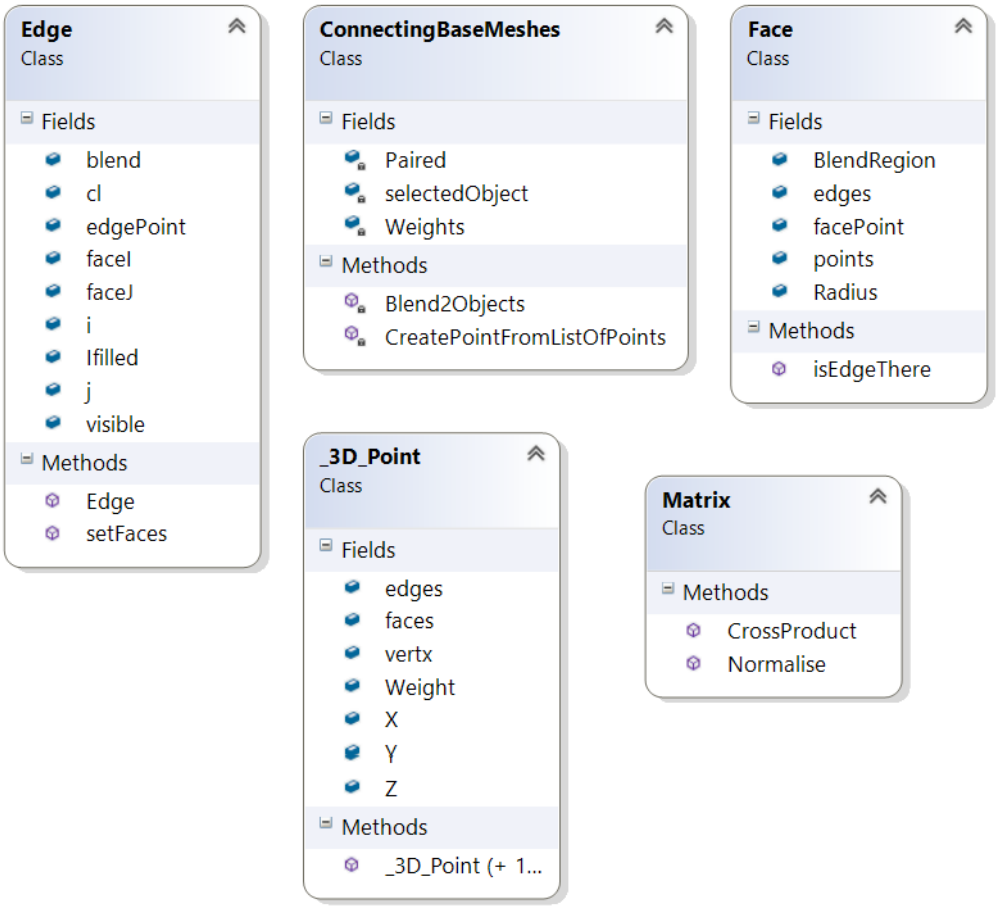
Values Weights	W1	W2	W3	W4	W5	W6
Weights (A)	0.5	0.5	0	0	0	0
Weights (B)	0.3	0.3	0.1	0.1	0.1	0.1
Weights (C)	0.4	0.4	0.05	0.05	0.05	0.05
Weights (D)	0.1666	0.1666	0.1666	0.1666	0.1666	0.1666
Weights (E)	0.2	0.2	0.15	0.15	0.15	0.15
Weights (F)	0.1	0.1	0.2	0.2	0.2	0.2

4.6.2: Connecting the blend curve with base meshes.

After creating the blend curve successfully, the connection between the first mesh and the blend curve must be established, as well as connecting the second mesh with the blend curve. Keeping in mind that this topology generates a valance of four in all the blend curve area. Several updates must be made for the new blended object.

- Adding both meshes vertices, polygons and edges.
- Adding the newly inserted points of the blend curve.
- Adding the edges that connect both meshes with the blend curve.
- Creating polygons from the blend curve area and adding them to the object.

4.6.3: Class Diagram



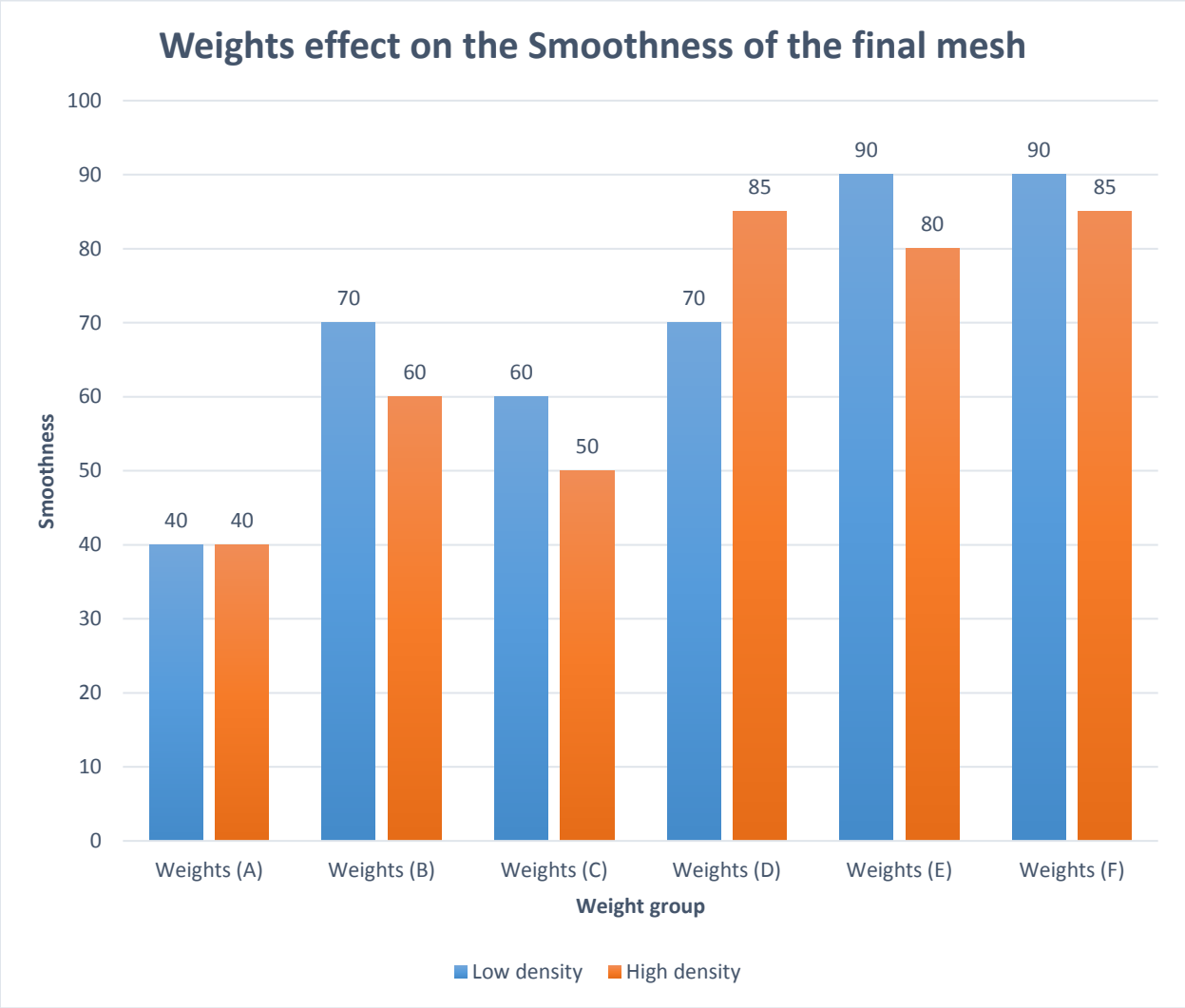
Chapter 5: Testing and Evaluation

Chapter 5: Testing and Evaluation

5.1: Testing

Having so many variables and attributes with different possible values affects the outcome of the entire project, in this section, multiple tests have been applied to showcase the best values for these variable/attributes.

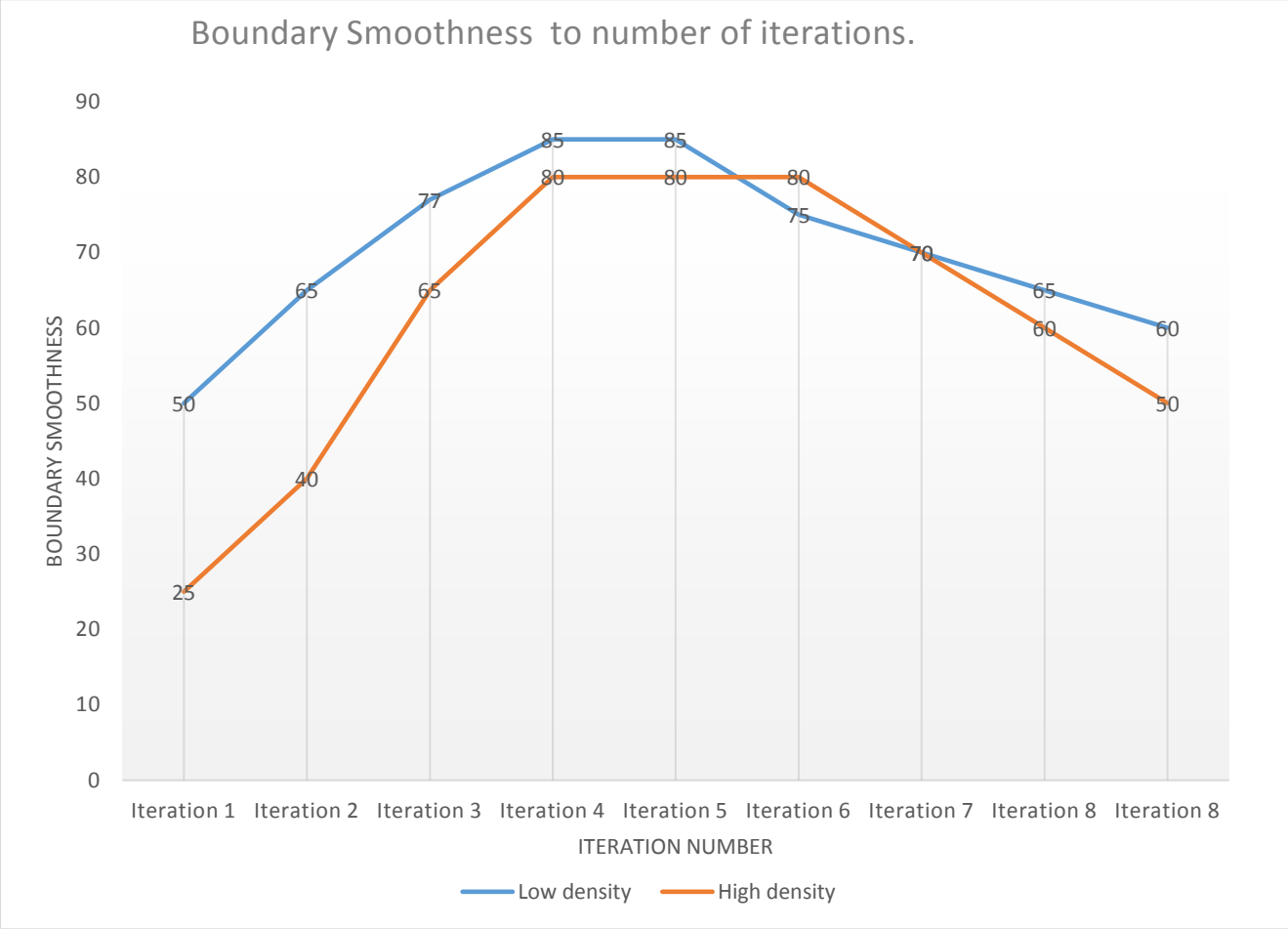
5.1.1: Weights effect on the smoothness of the final mesh



As mention before in section 4.6.1, the weights values could have multiple possibilities and these weights have been proven to generate great results, in the above graph, weight groups have different values and thus affect the final mesh completely differently as well, and each has a different outcome depending on the mesh density, wither if it is high or low. The weights group (F) have proven best amongst all the other groups due to its high smoothness percentage in both high and low density meshes.

Values Weights	W1	W2	W3	W4	W5	W6
Weights (A)	0.5	0.5	0	0	0	0
Weights (B)	0.3	0.3	0.1	0.1	0.1	0.1
Weights (C)	0.4	0.4	0.05	0.05	0.05	0.05
Weights (D)	0.1666	0.1666	0.1666	0.1666	0.1666	0.1666
Weights (E)	0.2	0.2	0.15	0.15	0.15	0.15
Weights (F)	0.1	0.1	0.2	0.2	0.2	0.2

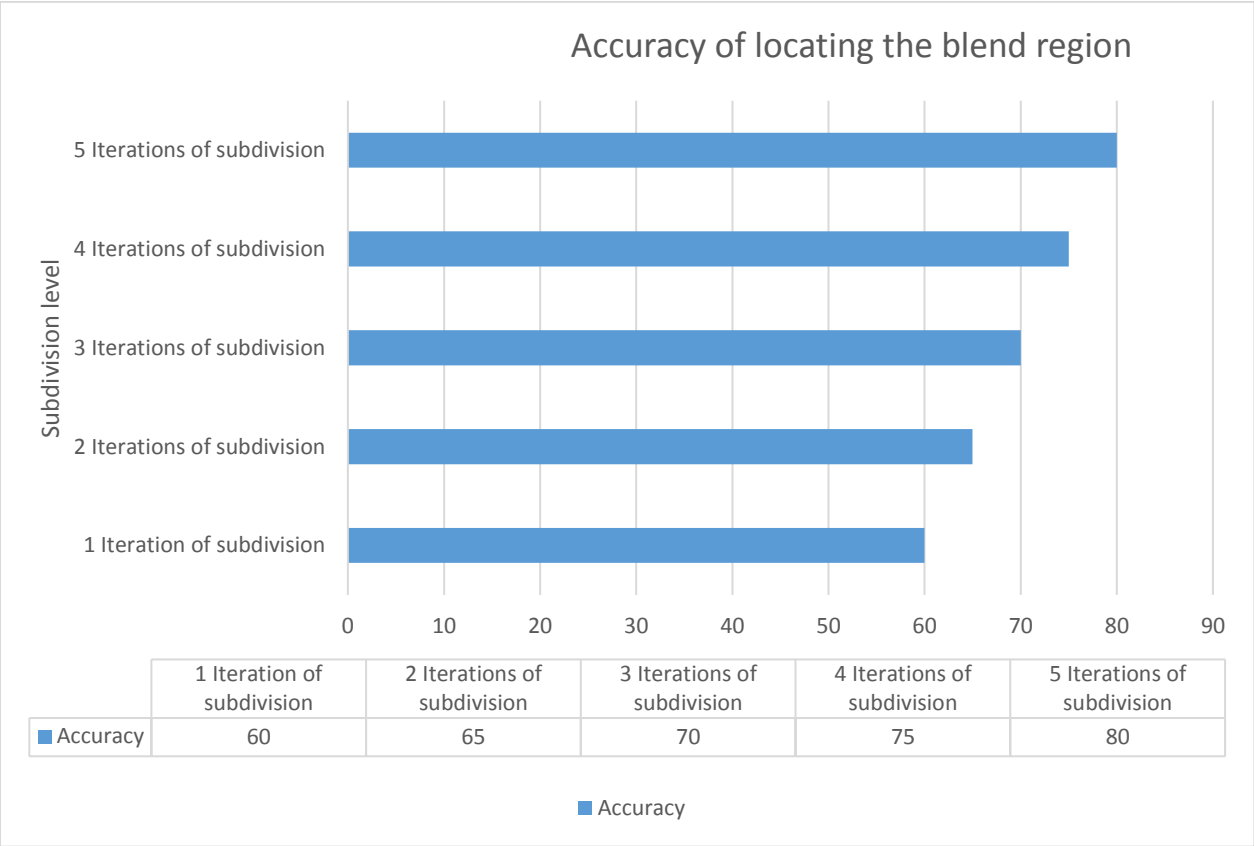
5.1.2: Boundary Smoothness to number of iterations



The boundary smoothing stage is a very important and critical stage in the blending process of two subdivision surfaces, and the process can be applied multiple times on the boundary curves to further smooth them.

The graph shows that the best number of iterations for smoothing the boundary curves is four iterations that is for both high and low density meshes. Any more iterations beyond would not increase the smoothness of the boundary curve, but rather decrease it and cause the points to move in a outwards direction and cause unwanted distortion.

5.1.3: Accuracy of locating the blend region



The accuracy of locating the blend region is affected by the subdivision levels of the two objects, the graph shows that the higher the subdivision level is, the better the results will be in terms of locating the blend region.

5.2: Evaluation

Stage	Criteria Of Evaluation	Expected Output	Stage Output
Sub-division	• Is the base mesh converted into a finer/smoother mesh? • Can the Sub-division process be applied again on the new mesh?	A finer smoother mesh that contains 4 times the number of polygons in the original mesh.	**100%**
Locating the blend region	• Did this stage successfully locate all possibly intersecting polygons?	A list of all intersecting polygons.	**100%**
Removing the blend region	• Did this stage successfully remove all the intersecting polygons from the data structure of the two meshes?	The two base meshes with the blend region removed.	**100%**
Boundary smoothing	• Did this stage successfully smooth the boundary points of the two meshes? • Can the boundary smoothing process be applied again on both meshes?	Smooth boundaries for the two intersected meshes.	**100%**
Matching vertices	• Did this stage successfully match the vertices in pair until there are no more pairs to be matched?	A list of paired vertices.	**100%**
Connecting base meshes	• Did this stage successfully create the blend region? • Did this stage successfully connect the two meshes with the blend curve? • Is the blend curve smooth?	The two meshes connected through the blend curve after adding the new inserted polygons.	**100%**

5.3: Complete case study

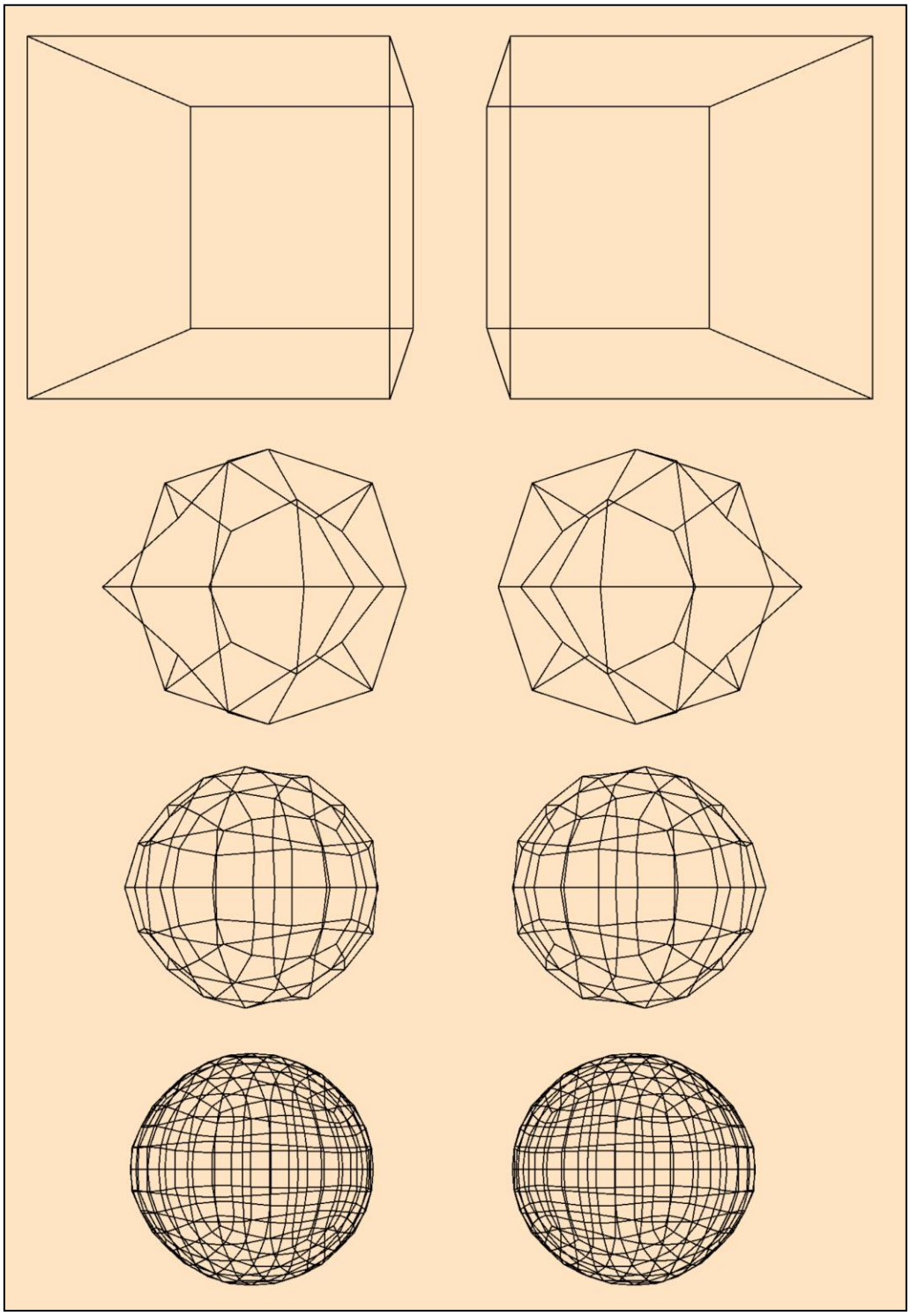


Fig. 37. Complete case study (1).

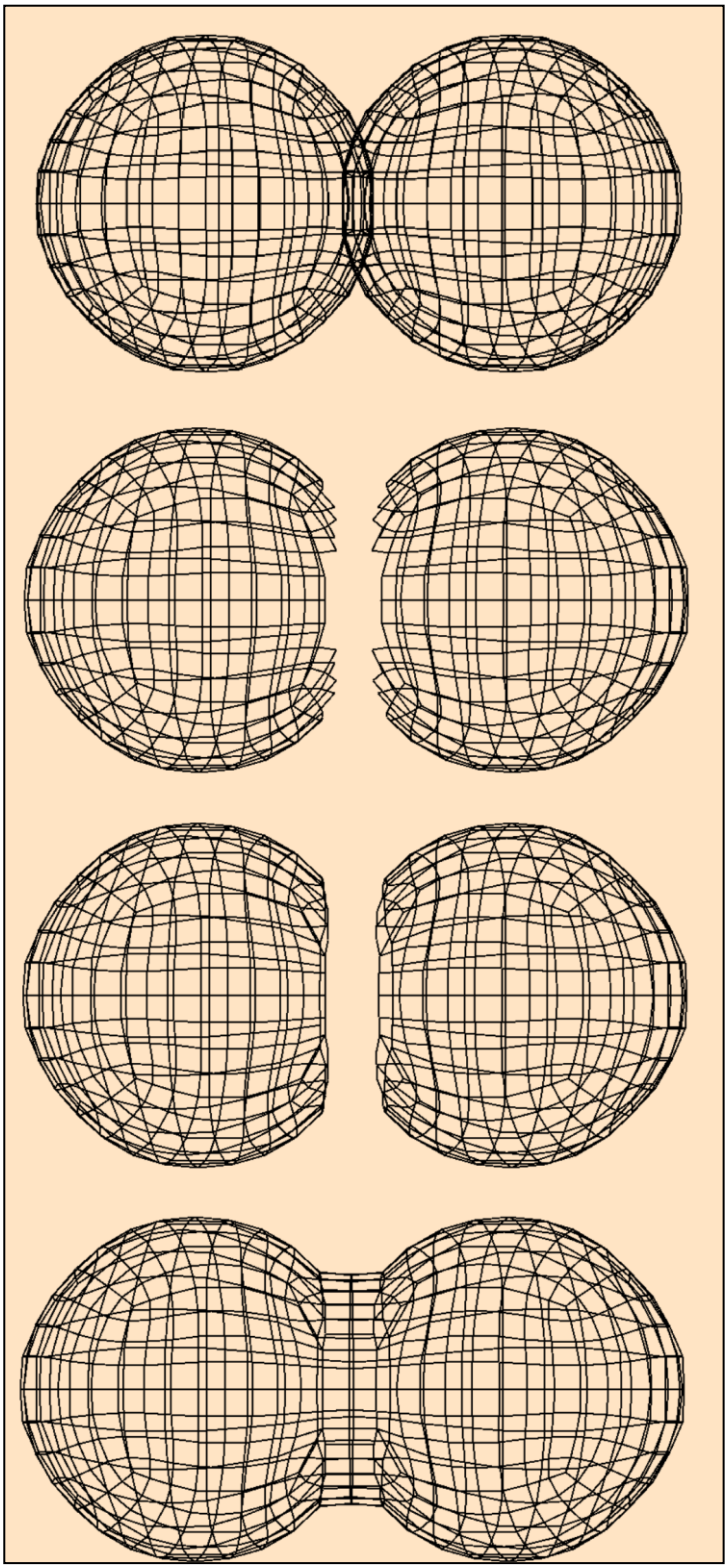


Fig. 38. Complete case study (2).

Chapter 6: Conclusion and future work

Chapter 6: Conclusion and future work

6.1: Final thoughts

To conclude, the smooth blending for subdivision surfaces as a method is a great tool for modeling complex freeform meshes while maintaining the smooth effect in the blending process. Other blending techniques use vertices on the intersecting curve as control points, which may result in unwanted distortion. That is due to the nature of the intersecting curve having sharp and extraordinary vertices, which in return generate irregular shapes in the region and cause unwanted distortion. The proposed algorithm on the other hand is very reliable, applicable and smooth. Working on this project has benefited me greatly in understanding different types of complex data structures, as well as the ability to understand and implement complex mathematical equations in 3D environment.

6.2: Challenges

6.2.1: Debugging challenges

Due to the nature of the project, various debugging errors have risen up in the implementation process, some of these errors/problems were easy to locate and solve, others were that much harder since we are dealing with hundreds if not thousands of points, edges and faces.

6.2.2: Technical challenges

Implementing the data structure for this project has been the hardest part, since the data structure is so huge due to the nature of the subdivision schemes, implementing it myself was definitely a challenge. Adding new vertices, edges or faces while constantly updating all the objects, creating new meshes from subdivision scheme using Catmull-Clark surface, removing some polygons while updating objects.

6.3: Future work

The field of 3D graphics in general and CAD in particular is a very wide and in constant development, so to keep up with the improvement one would have to implement various types of algorithms, techniques and methodologies, such as manipulation tools, which are widely used in CAD programs to manipulate, edit and develop meshes. Some examples of these tools are trimming, Boolean operations, off-setting, intersection and many more. As well as diving into 3D Morphing and deformation.

References

[1] Driscoll, R. (2008, 8 1). *CATMULL-CLARK SUBDIVISION: THE BASICS*. Retrieved from rorydriscoll.com: http://www.rorydriscoll.com/2008/08/01/catmull-clark-subdivision-the-basics/

[2] Fuhua Cheng, F. F. (2008). *Smooth Surface Reconstruction using Doo-Sabin Subdivision Surfaces.* University of Kentucky, Department of Computer Science. Lexington, KY 40506, USA: The Community for Technology Leaders.

[3] J. Bakker, P. B. (2018). *Smooth Blended Subdivision Shading.* Johann Bernoulli Institute, University of Groningen. Nijenborgh 9, 9747 AG Groningen, The Netherlands: Eurographics Proceedings.

[4] Jingjing Shen, J. K. (2014). *Conversion of trimmed NURBS surfaces to Catmull–Clark subdivision surfaces.* University of Cambridge, Computer Laboratory. Cambridge CB6 1DT, United Kingdom: Elsevier B.V.

[5] K.C Hui, Y. L. (2006). *Smooth blending of subdivision surfaces.* The Chinese University of Hong Kong, Computer-Aided Design Laboratory, Department of Automation and Computer-Aided Engineering. Shatin, Hong Kong.: Elsevier Ltd.

[6] Konečný, J. (n.d.). *Catmull–Clark Subdivision Surfaces.* Comenius University , Faculty of Mathematics, Physics and Informatics . Bratislava, Slovakia: cescg.

[7] Ma, W. (2004). *Subdivision surfaces for CAD.* City University of Hong Kong, Department of Manufacturing Engineering and Engineering Management. 83 Tat Chee Avenue, Kowloon, Hong Kong, China.: Elsevier Ltd.

[8] McGuire, M. (2000, 8 7). *The Half-Edge Data Structure*. Retrieved from Spring Conference on Computer Graphics: http://www.sccg.sk/~samuelcik/dgs/half_edge.pdf

[9] Weiyin Ma, H. W. (2009). *Loop subdivision surfaces interpolating B-spline curves.* City University of Hong Kong, Department of Manufacturing Engineering and Engineering Management. Hong Kong, China.: Elseier Ltd.

[10] Xumin Liu, Y. X. (2015). *A Research about Adaptive Subdivision Algorithm Based On DooSabin.* Capital Normal University, College of Information Engineering. y, Beijing 100048, P. R. China: International Journal of Signal Processing, Image Processing and Pattern Recognition.

[11] Zorin, D. (2006). *Constructing Curvature-continuous Surfaces by Blending.* New York: The Eurographics Association.