Sistemas & Telemática
ISSN: 1692-5238
EditorSyT@icesi.edu.co
Universidad ICESI
Colombia

Navarro Newball, Andrés Adolfo; Wyvill, Geoff; McCane, Brendan
Efficient Mesh Generation Using Subdivision Surfaces
Sistemas & Telemática, vol. 6, núm. 12, julio-diciembre, 2008, pp. 111-126
Universidad ICESI
Cali, Colombia

Available in: http://www.redalyc.org/articulo.oa?id=411534379004

Efficient Mesh Generation Using Subdivision Surfaces

Andrés Adolfo Navarro Newball
anavarro@puj.edu.co

Geoff Wyvill
geoff@otago.ac.nz

Brendan McCane
mccane@cs.otago.ac.nz

Fecha de recepción: 4-01-2008 Fecha de selección: 17-10-2008 Fecha de aceptación: 8-08-2008

RESUMEN

Las mallas poligonales y en particular las mallas triangulares son la estructura más utilizada para modelado en 3D. La estructura de datos 'bordes directos' es la forma más eficiente de representarlas y la subdivisión de superficies un modo adecuado de generarlas. Del estudio de subdivisión de superficies escogimos el método 'subdivisión √3' para la generación de mallas. Nuestro principal reto fue tomar ventaja de la estructura de datos 'bordes directos' encontrando fórmulas para una implementación eficiente. Decidimos utilizar archivos en el formato 3DS y convertirlos a 'bordes directos' para uso en nuestra aplicación. Probamos nuestro algoritmo con mallas de topología arbitraria y calculamos su eficiencia. Nuestra implementación será utilizada para la creación de la cabeza de un perro virtual.

PALABRAS CLAVE

Malla, Computación gráfica, Subdivisión de superficies

ABSTRACT

Polygonal meshes and particularly triangular meshes are the most used structure for 3D modelling. The 'direct edges' data structure is the most efficient way to represent them and subdivision surfaces is an appropri-

ate method to generate them. From a review of subdivision surfaces we chose the '√3 subdivision' method for mesh generation. Our main challenge was to take advantage of the direct edges data structure and to find the right formulas for an efficient implementation. We decided to use files in the 3DS file format and convert them to the direct edges data structures for use in our application. We tested our algorithm with arbitrary mesh topologies and calculated efficiency. Our implementation will be used in the creation of a virtual dog head.

KEY WORDS

Mesh, Computer graphics, Subdivision surfaces

Clasificación Colciencias: Tipo 1

1. INTRODUCTION

We needed an adequate method to represent the surface of a 3D virtual dog skull that will be part of a bigger project [1]. Our implementation of a parametric triangular mesh generation algorithm helped us to validate the technique and to confirm our choice.

A surface separates the interior from the exterior by a boundary. In order to generate a mathematical surface we need to establish continuity and neighbourhood relationships between samples. Complex shapes require piecewise representations and are split in sub-regions each one of which is defined by an individual function. These representations are just approximations of the surface and it can be demonstrated from the Taylor theorem that the approximation error in an interval **h** of the surface by a polynomial of degree **p** is $\mathbf{O(h^{p+1})}$. In order to decrease the error one can increase the degree of the polynomial or use more segments or sub-regions. Operations required by surfaces include evaluation, query and modification. Evaluation refers to the sampling of the surface and its attributes. Query aims to identify if a given point $\mathbf{p} \varepsilon \mathbf{R^3}$ is inside or outside the solid bounded by **S**. Modification refers to geometry changes such as surface deformations or topology changes. There are two major classes of representation. Parametric representations are defined by a vector valued parameterization function (1). Implicit or volumetric representations are defined by a zero set of a scalar valued function (2) [2]. Parametric representations are better for evaluation and modification, implicit representations are better for query [2].

$f: \Omega \rightarrow S$, where S is a surface, $\Omega \varepsilon R2$ and $S = f(\Omega) \varepsilon R3$. (1)

$F: R3 \rightarrow R$ where $S = \{ x \varepsilon R3 \mid F(x) = 0 \}$. (2)

Polygonal meshes approximate a surface by a mesh of planar polygonal facets. They are low complexity representations more efficient for rendering. In particular, triangular meshes remain the most used structure for 3D modelling. In [3] they argue that quadrilaterals are better than triangles capturing the symmetries of objects and that they are compatible with bi cubic patches used in commercial software. However, in [2] they state that triangles have become increasingly popular because they allow more flexible and efficient processing and avoid conversion errors. We decided to follow the approach from [2] and use triangular meshes. Additionally, we found that subdivision surfaces are an appropriate way to generate and create parametric polygonal meshes. They reduce the representation of a complex surface to a simpler control mesh that is able to generate refined surfaces.

2. METHODOLOGY

In order to find an appropriate representation, we reviewed literature on 3D modelling. From this, we identified the advantages of using triangular meshes. Also, we studied and created an algorithm to convert from the 3DS file format to the direct edges data structure which we found to be the most efficient one. Then, from the review of subdivision surfaces we chose the $\sqrt{3}$ subdivision method for generating the mesh. At this stage,

we had to find the right formulas for counting elements in the mesh (i.e. vertices and faces) in order to create an efficient implementation. We created and downloaded several input meshes in the 3DS format, converted them and tested our algorithm. The input meshes included closed regular surfaces, planes with boundaries, surfaces with non subdividable polygons and the arbitrary mesh of a spaceship. We calculated memory usage, performance and element growth. The implementation will be further used in the creation of the virtual dog head.

3. TRIANGULAR MESHES

Frequently, triangular meshes are considered as a coarse collection without a mathematical representation. Problems arising from a coarse mesh include little connectivity information (triangle soups), inconsistent normal orientations, gaps, intersecting patches and degenerate elements such as triangles with zero area. However, an acquired coarse mesh can be used as an input to produce an enhanced one (i.e. using subdivision surfaces) [2, 4, 5]. A parametric triangular mesh **M** can be described as **M (P, K)**, where P = Set of N positions $\mathbf{p}_i(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i) \in \mathbf{R}^3$ and **K** contains the description of the topology. Triangular meshes are the most simple and flexible continuous surface representation where only $\mathbf{C}^0$ continuity is required [2]. Here, complex surfaces are formed by triangular pieces with a linear parameterisation function with an approximation error of $\mathbf{O(h^2)}$, where **h** is the maximum edge length. The valence of a vertex is the number of vertices in its neighbourhood. In semi-regular triangular meshes, most of the vertices have a valence of 6. Vertices with a different valence are called extraordinary vertices. "An important topological quality of a surface is whether or not it is two-manifold, which is the case if for each point the surface is locally homeomorphic to a disk (or a half-disk at boundaries). A triangle mesh is two-manifold, if it does neither contain non-manifold edges or non-manifold vertices, nor self-intersections. A non-manifold edge has more than two incident triangles and a non-manifold vertex is generated by pinching two surface sheets together at that vertex, such that the vertex is incident to two fans of triangles", [2, p.19].

3.1. Data structures for triangular meshes.

Efficient data structures allow local and global traversal of a mesh. Operations include [2]:

- Access and enumeration of individual vertices, edges, faces.
- Oriented traversal of edges of a face (next edge in a face).
- Access to at least one face attached to a given vertex.

The direct edges data structure is the most efficient to deal with triangular meshes [2]. It is based on indices as references to each element where indexing follows rules that implicitly encode connectivity information (Table 1). Here, each edge is represented into two opposing halfedges consistently oriented counter clockwise (Figure 2). This data structure is only useful for triangular meshes and provides no explicit representation of edges (though it can be specialised to

Table 1. The direct edges data structure

Indices and operation	Description / calculation
f	Index of a face
Get halfedge (f, i) from face number	3f + i, with i ε {0, 1, 2} (3)
h	Index of a halfedge
Adjacent face from halfedge (h)	h/3 (4)
Next halfedge	(h+1) mod 3. (5)

other polygons). It groups the three halfedges belonging to a common triangle. Additional information is explicitly stored in arrays. For instance, for each vertex, the index of an outgoing halfedge is stored and; for each halfedge, the index of its opposite half edge and the index of the vertex the halfedge points to are stored. Boundaries are managed with special negative indices that indicate that the edge vertex is invalid.

4. SUBDIVISION SURFACES

Subdivision surfaces help in the creation and refinement of proper 3D mesh models. They are capable of representing an arbitrary geometrically unrestricted topology. They produce an efficient hierarchical structure and an object can be modelled as a low resolution control mesh from which we can generate new meshes by refining the previous one. Some subdivision methods such as Catmull and Clark [4] work on quadrilaterals or extend subdivision to a n-sided problem and are not restricted to triangles such as Doo and Sabin [5, 6]. Others such as butterfly [4], √3 Subdivision [6] and Loop [8] are specialised to triangles. [8] Has been widely used [9, 10, 11].

One can define: $\mathbf{M^0}$ $\mathbf{(P^0, K^0)}$ as the original coarse mesh (which can be used as the control mesh) and $\mathbf{M^j}$ $\mathbf{(P^j, K^j)}$ as the **j** times subdivided mesh. Here, $\mathbf{P^j}$ are the mesh points at level **j** of subdivision and $\mathbf{K^j}$ contains the description of the topology at level **j**. $\mathbf{M^j}$, with $j \to \infty$ is the approximation of a B spline limit surface. A subdivision scheme **S** takes the vertices from level **j** to level **j+1** so that $\mathbf{S(K^j) = K^{j+1}}$. A subdivision matrix or stencil **SM** maps a mesh $\mathbf{M^j}$ to a topologically equivalent refined mesh $\mathbf{M^{j+1}}$. Eigenvalues or characteristic roots are a special set of scalars associated with linear systems of equations such as matrix equations. In a subdivision scheme **S**, the eigenvalues of the subdivision matrix are important to determine if the method converges to a limit smooth surface. For instance, all subdivision schemes must guarantee adequate design of the **SM** stencil so that eigenvalues have a certain distribution and a continuous surface approximating the limit surface can be generated [7, 9]. In **SM,** every row is a rule to compute the position of a new vertex and every column tells how one old vertex contributes to the vertex positions in the refined mesh [2]. In stationary methods the refinement functions are the same for every subdivision level. However, different rules are applied to define sharp features, creases or to deal with extraordinary vertices [9]. Also, stop conditions can be used for adaptive refinement [7]. In some methods old vertices are repositioned for smoothing [7].

4.1. Subdivision techniques for triangular meshes

Butterfly subdivision [4, 9] converts each face into four using edge vertices. Artefacts and discontinuities are produced in vertices with valence different from 6. For example, it never produces smooth surfaces on extraordinary vertices and incorrect smooth regions can appear near high valence vertices due to eigenclustering (more than one eigenvalue per matrix). It is based on dyadic split (divide every edge in two) and does not produce $\mathbf{C}^2$ continuity. The modified butterfly deals with artefacts of the original butterfly [4]. It approximates the behaviour of a $\mathbf{C}^2$ surface based on dyadic split. Here, eigenclustering is avoided. Loop subdivision [9] is one of the simplest $\mathbf{C}^1$ subdivision schemes. Here, stencils are smaller and convergence rate is better than in modified butterfly. It is based on dyadic split and is not $\mathbf{C}^2$ at extraordinary vertices. The scheme does not look smooth for large valences, due to eigenclustering. The modified Loop [9, 12] increases the stencil in a minimal way around a vertex and avoids eigenclustering. Here, vertices have better structure at extraordinary points. However, ripples appear. The ternary Loop subdivision [13] uses three stencils. It achieves bounded curvature, manifold support, and convex Hull. It is $\mathbf{C}^2$ continuous. However, No rules have been defined to deal with boundaries or with sharp features.

√3 Subdivision [7] is a stationary subdivision scheme with slower topological refinement with trisection of every original edge (every two steps). It inserts a new vertex at the centre of every face (Figure 1a and 1b). Then, it creates the new faces (Figure 1c) and flips every original edge (Figure 1d). To do that, it uses simple stencils of minimum size and maximum symmetry. Here, a new vertex is calculated as the average of the three old ones (a new vertex only affects one face). Then, the old ones are relaxed using (6). The scheme uses a generation index to perform adaptive refinement and allows sharp feature lines. It is $\mathbf{C}^2$ continuous except at extraordinary points. All new vertices have valence 6 and the valence of the old ones is not changed. We chose this method because it produces more levels of subdivision with lower number of triangles and simpler rules.

$$S(p) = (1-\alpha_n)p + \alpha_n \frac{1}{n}\sum_{i=0}^{n-1} p_i \qquad (6)$$

Where

S(p) = relaxation stencil for vertex p.

$$\alpha_n = \frac{4 - 2\cos\left(\frac{2\pi}{n}\right)}{9}$$

n = valence

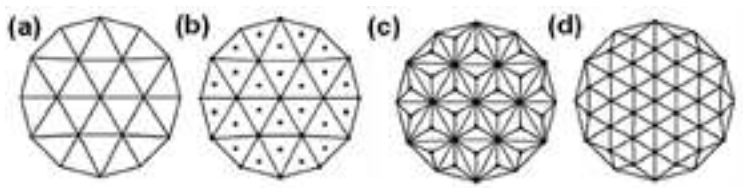

Figure 1. a) Original mesh. b) Middle point. c) Split triangles. d) Flipping edges. [adapted from 7, p.104]

5. IMPLEMENTATION

5.1. Converting from 3DS to Direct Edges

Using a 3DS format loader [14] we created an algorithm that converts a 3DS mesh into a Direct Edges data structure. In our algorithm, only the vertex list and faces descriptions were processed preventing unnecessary duplication. Our application opens the 3DS file. Next, it reads the vertices and adds them to a vertex list. Then, it reads the faces and adds them to a face list assigning the proper vertex indices. It creates the halfedges list from the faces list. For each one it assigns the vertex it points to and, for each vertex it assigns one outgoing halfedge. Next, for each halfedge it finds and assigns its opposite. Finally, it detects boundary vertices and saves the data.

5.2. √3 subdivison surfaces using the Direct Edges Data Structure

In our implementation a main subdivision cycle calculates the subdivision surface until it reaches the number of levels defined by the user. At each level, the algorithm performs three tasks. First, it calculates the new vertices in each face (Table 2). Here, if it is a non boundary subdividable face or a boundary face in even level (Figure 2c), the mid point is obtained and three faces are created. If the face is not subdividable then one face is recreated (Figure 2a). If the face is a boundary face and the level of subdivision is odd, two new vertices are calculated and three faces are created (Figure 2d). After the faces have been subdivided a mesh similar to the shown in Figure 1c is produced.

The next task (Table 2) is to flip the edges in each face of the new subdivided mesh. Here, the edges are flipped only if the face and its mate are sub dividable and non boundary or, when the face belongs to a boundary at an odd level of subdivision. After the edges have been flipped a mesh similar to the shown in Figure 1d is produced. The final task is to re- position the three old vertices in each face. Calculations are done taking care of avoiding boundary vertices, because these can cause visible discontinuities. Here, the calculation of the vertex's neighbourhood is required (Table 2).

5.2.1. Memory allocation

Every time the calculations for a new level of subdivision start, the memory is allocated according to the new needs. The result of the last subdivision step is stored in a file that is used as a starting point for the next level. The use of a file eliminated the need of using heavy intermediate memory objects. At level 0, memory for the control mesh is allocated according to the number of vertices, faces and halfedges which were defined as in Table 3. Memory for subsequent levels of subdivisions is calculated according to (7) (8) and (9).

$$NV = OVN + NSF + 2NBF \quad (7)$$

$$NF = (ONF - NNF)\ 3 + NNF \quad (8)$$

$$NHE = ((ONF - NNF).3.3) + 3NNF \quad (9)$$

NV = Number of vertices

NF = Number of faces

NHE = Number of halfedges. For each face 3 halfedges

OVN = Old number of vertices

Table 2. Steps in subdivision and neighbourhood calculation

Step1: finding new vertices	**Step2: flipping edges**
For AllFaces If FaceNotAtBoundary If Face Subdividable ComputeMidPoint and Split Else RecreateFace Else If FaceAtBoundary If EvenLevel If Subdividable ComputeMidPoint and Split Else RecreateFace Else If Subdividable CreateFacesAtBoundary Else RecreateFace	For AllFaces If FaceNotAtBoundary If Face Subdividable SwapFace Else RecreateFace Else If OddLevel If Subdividable SwapAtBoundary
Step 3: old vertices re positioning	**Neighbourhood calculation**
For AllFaces If VertexNotAtBoundary CalculateNeighbourhod RePosition Accoridng to (3)	Find OutgoingHalfedge AverageFirstVertex Increment Neighbourhood size Repeat Get OppositeHalfEdge Get NextHalfedgeInRing adding 1 Add vertex to average Increment Neighbourhood size Until RingComplete or Boundary or Distortion

Table 3. Halfedge elements implementation

Vertex (160 bits)	Float x, y, z Int Outgoing Int Boundary	3D position. For each vertex, store index of outgoing halfedge. Indicate if it is a boundary vertex.
Polygon (224 bits)	Int a, b, c, mate Int subdividable Int iBoundFace: Int iNonSubFaces	For each polygon store vertex indices, mate. Equals 1 when subdividable. Number of boundary faces before this. Number of non subdividable face before this.
Halfedge (64 bits)	Int oppositeHalfedge Int toVertex	For each halfedge store: Index of opposite halfedge. Index of vertex the halfedge points to.

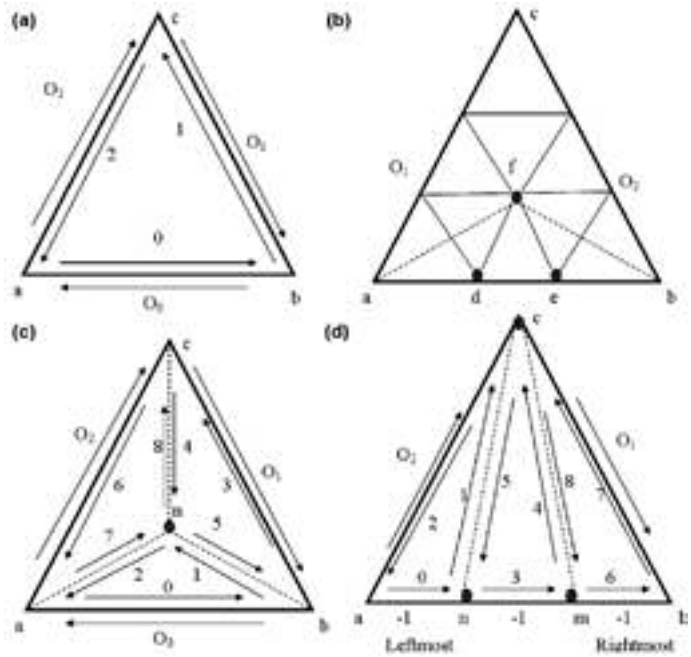

Figure 2. a) Non subdivided face. b) New vertices at boundary. c) Subdividing a regular face. d) Subdividing a boundary face in odd step.

NSF = Number of sub dividable faces. One new vertex for each face.

NBF = Number of boundary faces. Two new vertices for each face.

ONF = Old number of faces (each sub dividable one produce 3 new ones)

NNF = Number of non sub dividable faces.

5.2.2. Assigning halfedges and counting

Our main challenge was to keep track of the halfedges after each task. Assigning proper halfedges numbers is very important before the flipping operation, otherwise the mesh will be corrupted. The naive solution is to create costly loops to find and assign halfedge numbers. However, counting helps to use the direct edges data structure efficiently.

The first and easiest thing is to calculate the interior halfedges using formula (3). For example, a face **F** has three interior halfedges. These are assigned counter clock wise starting from vertex **a**. For instance:

- Halfedge 0 goes from a to b.
- Halfedge 1 goes from b to c.
- Halfedge 2 goes from c to a.

Figure 2a shows the halfedge numbers for face **F** = 0. In the regular case (Figure 2c) an old face produces 9 new halfedges. Assuming face [a, b, n] as face 0, [b, c, n] as face 1 and [c, a, n] as face 2, the halfedge values are easily obtained with (3). At boundaries (Figure 2d) faces are created differently. Here, we assumed the leftmost face as face number 0, the middle one as face 1 and the rightmost as face 2.

Let **G** be our face. Finding the values for the opposite halfedges for face **G** requires proper counting. Let us suppose that face **F** (before subdivision) is the neighbour face that contains the halfedge which is opposite to **G** (O_0, O_1, O_2 in Figure 2). The 0 halfedge of face **F** can be obtained from (10).

HE = ((F – N) 9) + 3N (10)

Where

HE = Halfedge number.

F = Current face number.

N = Number of non sub dividable faces before F.

However, we need to identify which halfedge from **F** we are next to. In order to do this, we need to know if the opposite (**O**) halfedge (O_0, O_1 or O_2) is the number 0, 1 or 2 in **F**. Here:

- If O = 3F then the O is F's number 0.
- If O = 3F+1 then the O is F's number 1.
- If O = 3F+2 then the O is F's number 2.

Once we have the **O** number from face **F,** as new halfedges will be created, we need to identify what will be the halfedge number after **F** has been subdivided. There are several cases:

- *F is a sub dividable face or F is a boundary face in even level of subdivision*. If **G** is opposite to **F**'s 0 halfedge, then O = HE. If **G** is opposite to **F**'s 1 halfedge, then O =HE+3. If **G** is opposite to **F**'s 2 halfedge, then O = HE+6.
- *F is a non sub dividable face.* If **G** is opposite to **F**'s 0 halfedge, then O = HE. If **G** is opposite to **F**'s 1 halfedge, then O = HE+1. If **G** is opposite to **F**'s 2 halfedge, then O = HE+2.
- *F is a sub dividable boundary face in odd level of subdivision.* Here, the new three triangles are arranged differently from the regular case (figures 2c and 2d), this affects the way we count. Here, if **G** is opposite to **F**'s 0 halfedge, then O = HE. If **G** is opposite to **F**'s 1 halfedge, then O = HE+6. If **G** is opposite to **F**'s 2 halfedge, then O = HE+2.

On the other hand, calculating the new interior opposite halfedges (1, 5, 4, 8, 7, 2 in Figure 2c and 1, 5, 4, 8 in Figure 2d) is straightforward. Here, halfedge 1 of one new face is next to halfedge 2 of the other one (11). Opposite halfedges to the 0 halfedge of new boundary faces are assigned -1 (non existent).

OppositeOf (3F+1) = (3(F+1))+2, with F = face number (11)

5.2.3. Splitting

A new vertex **n** in a regular face (Figure 2c) is obtained from (12). At boundaries (figures 2b and 2d), every odd subdivision level two new vertices are calculated in the edge of the boundary and the centre of the boundary triangle is displaced as in (13), (14) and (15)

n = 1/3 (a + b + c) (12)

d = 1/27(16a+f+10b) (13)

e =1/27(10a+f+16b) (14)

f = 1/27(4a+19f+4b) (15)

Sub triangles from [a, b, c] are built counter clock wise as [a, b, n], [b, c, n]

and [c, a, n] (Figure 2c), leaving their 0 halfedge to coincide with an old edge in the subdivided face. Sub triangles at a boundary face in an odd level of subdivision are arranged from left to right (Figure 2d). Here [a, b, c] is refined with [a, n, c], [n, m, c] and [m, b, c]. Note that the halfedge 0 of each new face is made to face the border.

5.2.4. Swapping faces

In Figure 3a, face **F** [a, b, c] is going to be swapped with face **M** [b, a, d]. Let us assume that **F** is a regular face. Swapping cannot be achieved if:

- **M** has already been swapped in the current subdivision level.
- **M** is a boundary face.
- **M** is a non sub dividable face.

If swapping is possible the old opposite halfedge values HE1, HE2, HE3 and HE4 have to be stored and reassigned in order to keep consistency and connectivity. Then, the two faces are swapped as shown with the dotted line and the new 0 halfedge is assigned along this line for each face in this new arrangement. The new faces are described in Table 4.

Table 4. Swapping faces F and M

F [d, c, a] halfedges		M [c, d, b] halfedges	
0: F3	Opposite: M3	0: M3	Opposite: F3
1: (F3) + 1	Opposite: HE1	1: (M3) + 1	Opposite: HE4
2: (F3) +2	Opposite: HE3	2: (M3) +2	Opposite: HE2

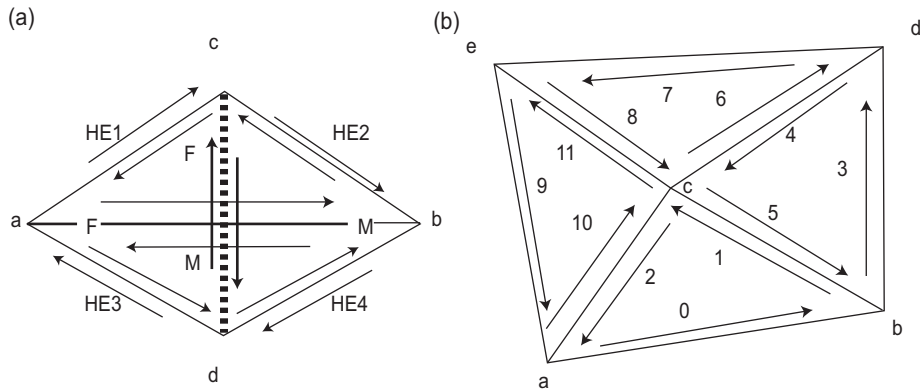

Figure 3. a) Flipping edges. b) Neighbourhood.

A boundary face in an odd level is swapped only if the same conditions for regular swapping are met (except that **M** could have been swapped); only the leftmost and rightmost faces (Figure 2d) are candidates for swapping, the middle face remains the same. Here (Figure 2b), flipping requires the rearrangement of triangles [a, f, O_1] and [a, d, f] to triangles [O_1, d, f] and [d, O_1, a] and triangles [f, b O_2] to [e, b, f] to triangles [e, O_2, f] and [O_2, e, b] producing a boundary strip of five triangles.

5.2.5. Detecting boundaries and subdividable faces

A boundary face is detected by checking if it contains at least one -1 valued opposite halfedge. Boundary vertices are detected as a last step in the conversion from the 3DS to the direct edges format. Here, a flag is assigned to each boundary vertex. In Figure 2a:

- **a** is a boundary vertex if the opposite of halfdegde 0 or 2 is next to a boundary face.
- **b** is a boundary vertex if the opposite of halfdegde 0 or 1 is next to a boundary face.
- **c** is a boundary vertex if the opposite of halfdegde 1 or 2 is next to a boundary face.

Not keeping proper track of the boundary vertices may produce distortion. The detection of a sub dividable face is yet to be implemented. Under certain flatness criteria (based on differential geometry) a face could be tagged as non sub dividable during the conversion process.

5.2.6. Calculating a neighbourhood

This calculation returns the number of neighbours of a vertex and their average coordinate value. In Figure 3b vertex **c** has 4 neighbours (a, b, e, d) and their average is obtained with (16). Our algorithm deals with closed manifolds, boundaries and distorted polygons. Let us assume that the first detected outgoing halfedge in Figure 3b is HE 2. First, we add vertex **a** to the average and increment the neighbourhood. Then, we find the opposite halfedge (in this case 10). If it is the 2 halfedge of the next face we subtract 2 in order to find the halfedge which points to the next vertex. If it is the 0 or 1 halfedge of the next face we add 1 in order to find the halfedge which points to the next vertex. Then, we add this vertex to the average and increment the number of neighbours. The algorithm continues until the halfedge pointing to the next vertex is equal to the first halfedge or until an edge or a distortion is detected (Table 2).

Average of **c**-neighbours = (a + b + e + d) / 4 (16)

6. RESULTS

Our direct edges implementation uses fewer elements. While in the 3DS format a cube contains 26* vertices, in the direct edges format it only contains 8. While in the 3DS format a spaceship contains 649* vertices, in the direct edges format it only contains 260. The number of faces remained the same in both cases. The size of one vertex, one polygon and one halfedge (Table 3) can be easily calculated as a float and an integer occupy 32 bits each. According to (7), (8) and (9) for a mesh object in a subdivision step the size in bytes would be (17). The intermediate text file generated in each subdivision will vary in size, but tests produced files with sizes lower than 4 kilo bytes. The complexity O(n) for the main

* As generated and verified using Discreet's 3D Studio Max

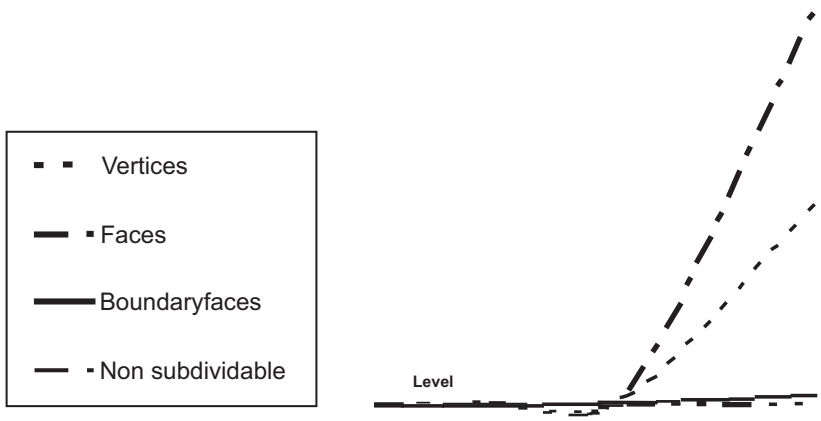

Figure 4. Element increase in √3 subdivision.

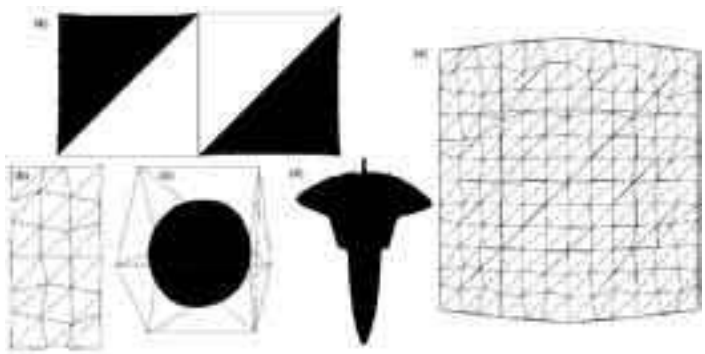

Figure 5. a) Mesh with boundaries and 2 non sub dividable faces (7 levels). b) 2D mesh with boundaries (2 levels). c) Limit and control surface of a cube (4 Levels). d) Spaceship (2 levels). e) Big plane (2 levels)

subdivision routine can be approximated as (18), because all values can be approximated in terms of **F** with the Euler formulas [2]. It accounts time for the three main procedures (splitting, flipping and re positioning) and the time to create and read from the intermediate file. The number of elements growth is shown in Figure 4. Here, it can be seen that the number of regular faces and the number vertices have an exponential growth, while the number of boundary faces growth is much slower and behaves linearly. The number of non sub dividable faces remains constant. Meshes produced are shown in Figure 5.

Size = (160NV + 224NF + 64NHE) / 8 (17)

O(n) = O (L F I) (18)

Where

L = Number of subdivision levels

F = Number of faces

I = Neighbourhood size (insignificant in big meshes)

7. DISCUSSION

Parametric triangular meshes allow efficient validation (used for display) and modification (especially when defined with subdivision surfaces). We decided to use them, because due to their popularity, many methods for processing them have been developed. Coarse meshes may bring distorted triangles and edges (i.e. "nonmanifold meshes are problematic for most algorithms, since around non-manifold configurations there exist no well-defined local geodesic neighbourhood"), [2, p.19]. We manage them with the definition of proper stop conditions. Our algorithm can be applied to an arbitrary mesh (Figure 5). The understanding of the direct edges data structure allowed efficient implementation and reduced the number of elements used in the original 3DS format. Counting was essential in memory allocation and in keeping track of the halfedges for each new face. It allowed the effective implementation of the basic operations required by the subdivision algorithm.

Subdivision surfaces have a major advantage, which is providing a control mesh (which will facilitate deformation) and a refined triangular mesh (which will ease display). Their use will bring $\mathbf{C^2}$ continuity to most regions of the mesh, producing a more natural look [2]. The benefits of the √3 subdivision brought to our attention a better and newer subdivision scheme which is supported by less complex stencils and has a simpler way to deal with sharp features, boundaries and with adaptive refinement. In √3 subdivision growth is slower than in other subdivision techniques. The inclusion of non sub dividable faces lowers the number of new generated faces. Apart from boundary faces, the growth is still exponential (Figure 3). However, the √3 subdivision produces fewer triangles than other methods providing similar quality and the mesh produced is visually appealing with a few levels of subdivision.

One simple way to define edges or sharp features is to pre assign -1 valued opposite halfedges to the region of interest. We want to propose preserving flat regions with smaller number of triangles. For instance, differently from the original algorithm we want to avoid subdividing well defined flat regions during adaptive refinement. To achieve this, our algorithm still requires a flatness function able to predetermine what faces are non sub dividable.

Londra [1] will be a virtual dog capable of displaying facial expressions. She will use a skull model implemented with a polygonal mesh generation / representation technique suitable for deformation, parameterization and animation, because the skull shape will change according to conformational anatomical parameters. From our implementation and the tests made, we have found that √3 subdivision surfaces are an appropriate modelling technique for the skull. Additionally, our skull will include non penetration features, requiring the use of an alternate implicit model

(implicit models are better for query operations). Here, the idea will be to create new dog breed heads from new skull shapes obtained altering the model through its control mesh that will produce the new refined meshes. To account for head type transitions, Londra will not require topological changes, only geometric ones. In consequence, the complexities involved in the former will be avoided.

8. BIBLIOGRAPHIC

[1] Navarro AA, Wyvill G, McCane B. Towards the creation of Londra: A virtual expressive animated dog. Accepted for NZCSRSC, 2008.

[2] Botsch M., Pauly M., Kobbelt L., Alliez P., Lévy B. Geometric Modeling Based on Polygonal Meshes. SIGRAPH Courses, 2007.

[3] DeRose T, Kass M, Truong T. Subdivision Surfaces in Character Animation. Annual conference on Computer graphics and interactive techniques Proceedings, pp. 85 – 94, 1998.

[4] Watt A, Policarpo F. 3D Games: Real time rendering and software technology. Addison – Wesley, London, 2001.

[5] Doo D. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. Interactive Techniques in Computer Aided Design Proceedings, pp. 157 - 165, 1978.

[6] Doo D, Sabin M. Behavior of recursive division surfaces near extraordinary points, Computer-Aided Design, 10(6)356-360 (1978)

[7] Kobbelt L. √3 Subdivision. Annual conference on Computer graphics and interactive techniques Proceedings, pp. 103 – 112, 2000.

[8] C. T. Loop. Smooth Subdivision Surfaces Based on Triangles. M.S. Thesis, University of Utah, 1987.

[9] Zorin, Stationary Subdivision and Multiresolution surface representations. PhD Thesis. California Institute of Technology, 1998.

[10] Lee A, Sweldens W, Schröder P, Cowsar L, Dobkin D. Multiresolution Adaptive Parameterization of Surfaces. Annual conference on Computer graphics and interactive techniques Proceedings, pp. 343 – 350,1999

[11] Lee A, Moreton H, Hoppe H. Displaced subdivision surfaces. Annual conference on Computer graphics and interactive techniques Proceedings, pp. 85 – 94, 2000.

[12] Stam J. Evaluation of Loop Subdivision Surfaces, SIGGRAPH Computer Graphics Proceedings, 1998.

[13] Loop C. Smooth Ternary Subdivision. Curve and Surface Fitting: Saint-Malo 2002.

[14] Vitulli D. The 3DS file structure. http://www.spacesimulator.net, 2005.

CURRÍCULUM

Andrés A. Navarro Newball. Computer Scientist from the Universidad Javeriana. MSc in Computer Graphics from the University of Hull. Networks Specialist from

the ICESI University. Beneficiary of the Coimbra Scholarship at the Universitá degli Studi di Siena. He was part of the Colombian Telemedicine Centre. He is lecturer at the ICESI and the Universidad Javeriana where he leads the DESTINO research group. He is a PhD candidate at the University of Otago.

Dr. Geoff Wyvill. BA in Physics from Oxford, MSc and PhD in Computer Science from Bradford. Professor at the Department of Computer Science at the University of Otago. He directs the Computer Graphics lab at the Computer Science Department. Teaching over 30 years, he accounts over 100 publications in Computer Graphics and is member of several editorial boards.

Dr. Brendan McCane. Received his BSc(Hons) and PhD from James Cook University of North Queensland. He joined the Department of Computer Science at the University of Otago in 1997 and has been Head of Department since 2007. His research interests include computer vision, pattern recognition, medical imaging, machine learning and computer graphics.