EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

Pixar's semi-sharp creases

M. Babtist[1] and S. Wehkamp[1]

[1]Rijksuniversiteit Groningen, Netherlands

Abstract

Until 1998 the modelling of believable and endearing characters in computer graphics was done using NURBS surfaces despite the severe topological restrictions they impose. In order to remove these restrictions subdivision surfaces could be used. However the usage of subdivision surfaces in animation was restricted due to some difficulties. One of these difficulties is the modelling of semi-sharp creases. Here we describe what methods are required on how to overcome this difficulty. This includes background information on the topic and an implementation of the method.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Boundary representations I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

In 1998 one of the most common ways to model complex smooth surfaces was by using a framework of Non-Uniform Rational Basis Splines (NURBS). These were mainly used because they already existed in popular commercial systems. However during the animation of Toy Story 1 Pixar experienced several difficulties with these surfaces. A couple of Pixar employees decided to try and find a solution to these difficulties.

An existing solution for their problems already existed in the form of subdivision surfaces. These surfaces do not require any trimming and their smoothness is guaranteed. Although these surfaces had been used before this paper [DKT98], the usage was not yet widespread. This is mostly because systems have been developed to add features to NURBS models which do not work by default on subdivision models. An example of such a feature is variable radius fillets: corners with a variable sharpness. To implement this the main idea is to generalise infinitely sharp creases in order to obtain creases whose sharpness can vary from zero (smooth) to infinite.

Infinitely sharp creases are convenient for representing piecewise-smooth surfaces. However in the real world surfaces are not infinitely sharp but are always smooth when viewed sufficiently close. For animation we would like to capture these tightly curved shapes.

2. Problem definition

A NURBS surface is limited to representing a sheet, a cylinder, or a torus. This is a limitation for any surface that imposes a global planar parameterization. A single subdivision surface does not have this limitation and can represent surfaces of arbitrary topology.

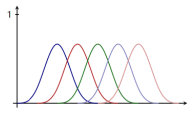
2.1. Non Uniform Rational B-splines

In 1998 the most common way to represent complex surfaces was using NURBS. A B-spline or *basis spline* refers to a class of basis functions which can be linearly combined to form a smooth spline function. B-splines are piece wise polynomial: they consist of separate section of polynomial joined together at positions called *knots*. These joins are constructed so that they are as smooth as possible. Degree d B-splines have $d-1$ continuous derivatives across each knot. Control points are used to control the shape of the curve [Cas10].

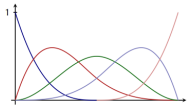
The collection of knots for a B-spline is called the knot vector. In a *uniform* B-spline the knots are equally spaced while in a *non-uniform* knot vector the knots are arbitrarily positioned. An example of a uniform basis function can be found in Figure 1a where the resulting curve is shown in Figure 2a. Using the same control points but with the different non-uniform knot vector shown in Figure 1b the curve shown in Figure 2b can be obtained [Cas10].

NURBS surfaces however suffer from a couple of difficulties:

1. Trimming is expensive and prone to error.
2. It is difficult to maintain smoothness, or even approximate smoothness, at the seams of the patchwork as the model is animated. The authors of [DKT98] use the creation of the character *Woody* as an example. A considerable amount of manual effort was required to hide the seams in the face of *Woody*.
3. Local refinement of NURBS surfaces is difficult since an entire control point row, column, or both must be added to preserve the grid structure



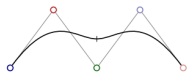
(a) The uniform case. Each basis function is an identical translated copy of the others.



(b) An example of the non-uniform case. In this example every basis function is unique in shape.

Figure 1: Comparison of uniform and non-uniform basis function [Cas10].

(a) Cubic B-spline using the uniform basis functions of Figure 1a.



(b) Cubic B-spline using the non-uniform basis functions of Figure 1b.

Figure 2: Comparison of uniform and non-uniform cases [Cas10].

2.2. Subdivision

There are various subdivision methods available which can be classified based on four criteria [ZP00].

Refinement The refinement type can either be face split or vertex split. Face split based subdivision splits a face into multiple smaller faces as can be seen in Figure 3. Vertex split based algorithms on the other hand subdivide a vertex into multiple smaller vertices as can be seen in Figure 4.

Mesh type The algorithms either work on triangular meshes or quadrilateral meshes.

Approximation vs. Interpolation Face split schemes can either be interpolating or approximating. With interpolation the vertices of the coarser tiling are also vertices in the refined tiling. This is useful because the control points in the original tiling are also points in the limit surface which allows one to control it in an intuitive manner. Another advantage is that algorithms can be simplified and many calculations can be performed "in place".

	Triangular meshes	*Quad. meshes*
Approximation	Loop (C^2)	Catmull-Clark (C^2)
Interpolation	Mod. Butterfly (C^1)	Kobbelt (C^1)

Table 1: The various face split subdivision methods which are available [ZP00].

Doo-Sabin, Midedge (C^1)
Biquartic (C^2)

Table 2: The various vertex split subdivision methods which are available [ZP00].

A disadvantage of these surfaces however is their quality which is lower than surface produced by approximating schemes. Surfaces created by interpolating schemes also do not converge as fast to the limit surface as approximating schemes.

Smoothness The last distinction which can be made is the quality of the limit surface which can for example be C^1, C^2, etc.

A final overview of the various subdivision methods can be found in Table 1 where the face split subdivision methods are listed and in Table 2 where the vertex split subdivision methods are listed [ZP00].

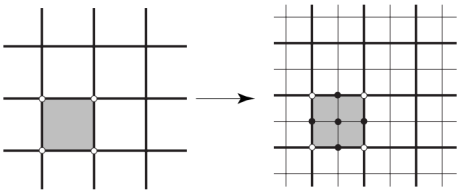


Figure 3: Face split for quads [ZP00].

In the sections below two subdivision schemes are discussed which were available at the time of the original paper.

2.3. Loop Scheme

The Loop scheme is an approximating face split scheme used for triangular meshes and was proposed by Charles Loop in 1987 [Loo87]. Since Loop subdivision is an approximating scheme we compute new (odd) and the original (even) vertices. One iteration of the algorithm would look like this:

1. Create an adjacency data structure
2. Compute odd vertices using the rules shown in Figure 5a.
3. Compute the even vertices with the rules shown in Figure 5b.
4. Connect the vertices to create the new faces.

The final scheme produces C^2-continuous surfaces over regular meshes.

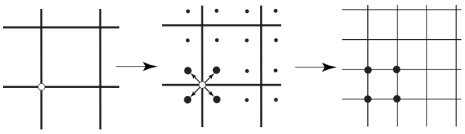
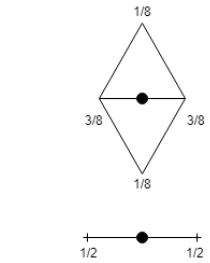
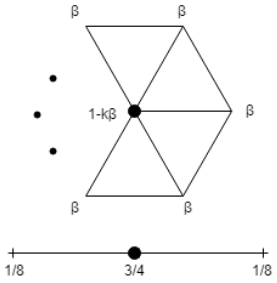


Figure 4: Vertex split for quads [ZP00].



(a) Masks for odd vertices.



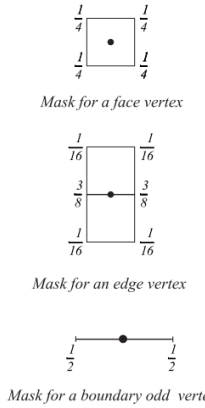
(b) Masks for even vertices.

Figure 5: Rules used by Loop subdivision. β can be chosen to be either $\frac{1}{n}(5/8-(\frac{3}{8}+\frac{1}{4}\cos(\frac{2\pi}{n}))^2)$ for the original choice of Loop or for $n>3$, $\beta=\frac{3}{8n}$ as proposed by Warren. For $n=3$, $\beta=\frac{3}{16}$ can be used [ZP00].

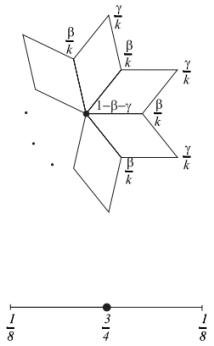
2.4. Catmull-Clark Scheme

The Catmull-Clark scheme is another approximating face split scheme like the Loop scheme, but Catmull-Clark is used on quadrilateral meshes instead of triangular meshes. It was proposed by E. Catmull and J. Clark in 1978 [CC78]. The rules for the Catmull-Clark scheme can be found in Figure 6

Infinitely sharp creases are very convenient for representing piecewise-smooth surfaces. However, real world surfaces are never infinitely sharp. All real world corners are smooth when viewed sufficiently close. In animation we would like to capture these tight curves. This however is very difficult using Catmull-Clark, requiring precise positioning of many tiny quads, since the tight curves would be smoothed out otherwise.



(a) Masks for odd vertices.



(b) Masks for even vertices.

Figure 6: Catmull-Clark subdivision rules. Catmull and Clark [CC78] suggest to use $\beta=\frac{3}{2k}$ and $\gamma=\frac{1}{4k}$ [ZP00].

3. Solution method

As discussed in the previous section several subdivision schemes are available. The authors of the paper [DKT98] chose to base their work on an approximation scheme since the quality of those schemes is higher. They chose the Catmull-Clark subdivision scheme over Loop subdivision because of two reasons:

1. "They strictly generalise uniform tensor product cubic B-splines, making them easier to use in conjunction with existing in-house and commercial software systems such as AliasWavefront and SoftImage." [DKT98]
2. "Quadrilaterals are often better than triangles at capturing the symmetries of natural and man-made objects. Tube-like surfaces like: arms, legs, and fingers. These objects can be modelled much more naturally with quadrilateral." [DKT98]

To achieve semi sharp creases using Catmull-Clark subdivision the authors propose a method called *hybrid subdivision*. The idea is to use one set of rules a finite but arbitrary number of subdivision steps, followed by another set of rules that are applied to the limit. The first set is used to make the crease sharp while the second set of rules is used to guarantee the smoothness. The sharp rules are the same as the boundary rules shown in Figure 6. This results in semi-sharp creases which are sharp at coarse scales and smooth at finer scales. Intuitively these rules can be seen as that the new spline only depends on vertices along the crease. Two cases can be distinguished:

Case 1 The sharpness is a constant integer s, where $s>0$. We subdivide s times using the infinitely sharp rules after which the smooth rules are applied. The two subedges created have a sharpness of $s-1$. A sharpness of $s=0$ is considered smooth. If $s\to\infty$ the sharp rules are used for all steps thus leading to an infinitely sharp edge.

Case 2 A constant but non integer sharpness s. The main idea here is that we interpolate between the adjacent integer sharpness. Let $s\uparrow$ and $s\downarrow$ be the floor and ceiling of s respectively.
The first version of the crease is obtained by subdividing $s\downarrow$ times using the sharp rules and then subdividing one additional time using the smooth rules. The second version of the crease is obtained by subdividing $s\uparrow$ times using the sharp rules. The final vertex positions are obtained by linear interpolation between both versions of the crease. The formula for the final vertex position can be found using:

$$v_i = (1-\sigma)v\downarrow_i + \sigma v\uparrow_i$$

where $\sigma = (s - s\downarrow)/(s\uparrow - s\downarrow)$.

In order to allow for non-constant, varying sharpness along a crease we need to generalise the two cases listed above. We associate sharpness per edge rather than per vertex since there is no single sharpness that can be assigned to a vertex where two or more creases cross.

3.1. Semi-sharp creases in Loop subdivision

The paper [DKT98] focuses on semi-sharp creases in combination with Catmull-Clark subdivision. However it would also be useful to have semi-sharp creases in Loop subdivision. It is possible to adjust the rules to generate C^0 continuous Loop subdivision surfaces. Similar to the Catmull-Clark implementation we again define a sharpness s and the number of incident sharp edges is vk. Now we define the Loop subdivision rules as follows:

Vertex Point Rules A vertex can classified into one of the types listed in Table 3 using s and vk. The masks are shown in Figure 7.

Edge Point Rules The edge point evaluation rules are determined according to the type of its two end vertices as can be seen in Table 4. Subdividing a sharp edge will generate two sharp subedges defined as the parents sharpness minus one similar to the Catmull-Clarks semi-sharp creases. If $0 \leq 1.0$ the edge point should be evaluated in the same way as in Catmull-Clark:

$$v_i = (1-\sigma)v\downarrow_i + \sigma v\uparrow_i$$

where $\sigma = (s - s\downarrow)/(s\uparrow - s\downarrow)$ [HFN*14].

4. Implementation

Our implementation is based on an existing Catmull-Clark implementation, adding the rules specified in Section 3. Put simply, the changes made to the basic implementation are as follows:

1. Add a sharpness value to the representation of edges.
2. Provide a way to change the sharpness value of (an) edge(s).

vk	s	type
< 2		smooth rule
$= 2$	≥ 1.0	crease rule
$= 2$	$[0.0, 1.0)$	$(1-vs) * v_s mooth + s \cdot v_c rease$
> 2	≥ 1.0	corner rule
> 2	$[0.0, 1.0)$	$(1-vs) * v_s mooth + s \cdot v_c orner$

Table 3: Vertex type classification using vk and s [HFN*14].

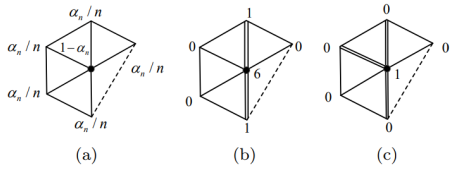


Figure 7: smooth (a), crease (b), and corner (c) rules of evaluating a vertex point. Double lines denote sharp edges [HFN*14].

3. Change vertex subdivision rules to use the sharpness rules where appropriate.
4. Change edge subdivision rules to use the sharpness rules where appropriate.

Change 1 is simple and trivial, and we will not go into detail. Keep in mind, however, that the sharpness must be decreased by 1 in every subdivision step (to a limit of 0).

4.1. Change sharpness value of edges

In order to actually show the differences caused by the rules as described in Section 3, we must have a way to change the sharpness of edges. In our implementation, we chose to have edge selection; by clicking the screen, the nearest (control net) edge will be toggled. One can select multiple edges this way and set their sharpness together. This involves keeping track of the control net separately from the current mesh status, along with the selection status of the edges.

	smooth/ dart	regular crease	irregular crease	Corner
smooth/ dart	(a)	(a)	(a)	(a)
regular crease	(a)	(b)	(c)	(c)
irregular crease	(a)	(c)	(c)	(b)
corner	(a)	(c)	(b)	(b)

Table 4: Edge type classification. Masks are shown in Figure 8 [HFN*14].

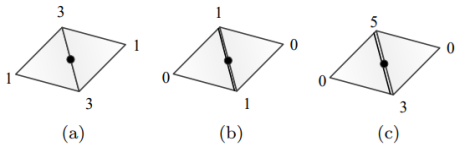


Figure 8: smooth (a), crease (b), and irregular crease (c) rules of evaluating an edge point. Double lines denote sharp edges [HFN*14].

The actual selection mechanism works as follows: whenever the user clicks the display area, a ray is cast from the point the user clicked, straight away from the screen and into the display. The edge that is closest to this ray is then selected. For the algorithm used, see algorithm **??**.

Data: Mesh m, Ray r, Model View Matrix *modelView*
Result: The selected HalfEdge
$p_2 \leftarrow$ last column of inverted *modelView*;
minDist $\leftarrow 1000$;
for *all HalfEdge* e_i *in* m **do**
  $d \leftarrow$ direction of e_i;
  $n_1 \leftarrow ray \times d$;
  $n_2 \leftarrow n_1 \times ray$;
  $p_1 \leftarrow$ coords of target of e_1;
  $linePercent \leftarrow ((p_2 - p_1) \cdot n_2)/(d \cdot n_2)$;
  if *linePercent in* $(0,1)$ **then**
    $dist \leftarrow |n \cdot (p_2 - p_1)|$;
    if $dist < minDist$ **then**
      $minDist \leftarrow dist$;
      $selectedEdge \leftarrow e_i$;
    end
  end
end
return *selectedEdge*;

Algorithm 1: Edge selection pseudocode

4.2. Vertex subdivision

For the vertex subdivision rules, we insert our code just before the standard Catmull-Clark rules. We loop over all edges connected to the current vertex, checking their sharpness, as well as keeping track of how many sharp edges we are dealing with. We also save the first two sharp edges we come across.

Once this is done, we check to see how many sharp edges there are. If there are 0 or 1, we use the normal Catmull-Clark rules. If there are more than 2, we simply return the position we already have; this vertex is a corner. If there are exactly 2, we use the two previously saved edges to get the other vertices, add them together, and add 6 times the current coords, then divide by 8. This is according to the rules in Figure 6b.

If the sharpness we are dealing with is in $(0,1)$, we use the interpolation formula as described in case 2 of Section 3, with $v\downarrow_i$= the previous result and $v\uparrow_i$= the normal Catmull-Clark version. The algorithm is provided in algorithm **??**.

Data: HalfEdge e
Result: Vertex coordinates
sharpEdgeCount $\leftarrow 0$;
$v \leftarrow$ target of e;
for *all* e_i *targeting* v **do**
  if *sharpness of* $e_i > 0.0$ **then**
    sharpness $\leftarrow$ sharpness of e_i;
    $sharpEdgeCount \leftarrow sharpEdgeCount + 1$;
    if *sharpEdge*1 *is empty* **then**
      $sharpEdge1 \leftarrow e_i$;
    else if *sharpEdge*2 *is empty* **then**
      $sharpEdge2 \leftarrow e_i$;
    else
      break;
    end
  end
end
if $sharpEdgeCount > 2$ **then**
  return *coords of* v;
else if $sharpEdgeCount < 2$ **then**
  return *CatmullClarkVertexRules*(v);
else
  $vertexPoint \leftarrow$coords of target of $sharpEdge1$;
  $vertexPoint \leftarrow vertexPoint$ + coords of target of $sharpEdge2$;
  $vertexPoint \leftarrow vertexPoint + 6\times$ coords of v;
  $vertexPoint \leftarrow vertexPoint/8$;
  if $sharpness < 1.0$ **then**
    $vertexPoint \leftarrow vertexPoint \times sharpness$;
    $vertexPoint \leftarrow vertexPoint + (1 - sharpness) \times CatmullClarkVertexRules(v)$;
  end
  return *vertexPoint*;
end

Algorithm 2: Vertex rules pseudocode

4.3. Edge subdivision

For the edge subdivision rules, we — like above — insert our code just before the standard rules. We check our current edge to see if it has an associated sharpness value. If this value is above 1.0, we use the Catmull-Clark boundary rules that already exist in the code. If it is 0, we use the Catmull-Clark inner rules. If it is between the two, we interpolate between the two, as before. The algorithm is provided in algorithm **??**.

Data: HalfEdge e
Result: Vertex coordinates
if *sharpness of* $e > 1.0$ **then**
  return $CatmullClarkBoundaryEdgeRules(e)$;
else if *sharpness of* $e > 0.0$ **then**
  return *sharpness of*
  $e \times CatmullClarkBoundaryEdgeRules(e) +$
  $(1 - $*sharpness of*
  $e) \times CatmullClarkInnerEdgeRules(e)$;
else
  return $CatmullClarkEdgeRules(e)$;
end

Algorithm 3: Edge rules pseudocode

5. Results

In this section, we will discuss the results of the proposed methods and its implementation.

As you can see in Figure 9 and Figure 10, our implementation does its job in demonstrating the effects of the algorithm. The marked edges become faintly creased with a low sharpness, and sharp with a high sharpness.

Pixar went on to use these techniques in various projects, including *Geri's Game* as an example project, and *Toy Story 2*. A sample from *Geri's Game* can be seen in Figure 11. Pay close attention to the details on the piece, as well as Geri's fingernails.

A comparison between the old NURBS based method and the method covered here can be seen in Figure 12.

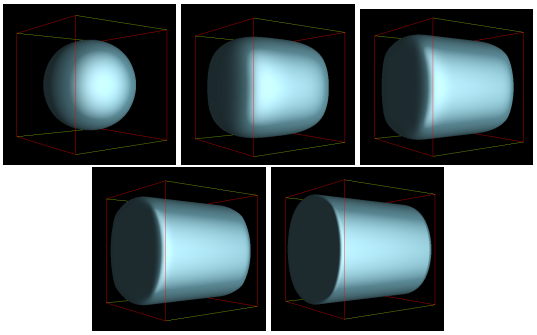


Figure 9: Results of the implementation on a cube mesh. The red edges have a sharpness of 0, 1, 2, 3, and 10 respectively, and subdivision has been applied 5 times.

6. Current situation

This section will discuss some more recent developments in the area of Catmull-Clark subdivision focusing on semi-sharp creases.

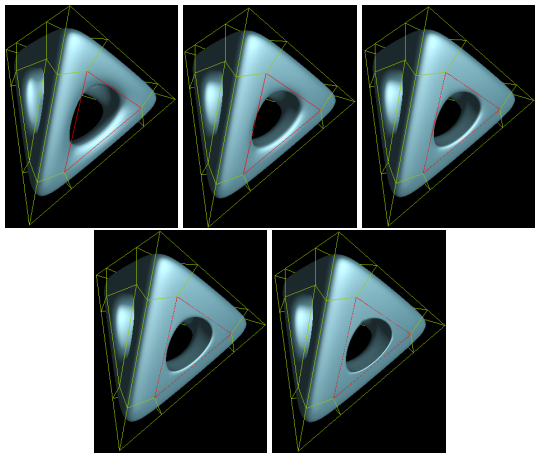


Figure 10: Results of the implementation on a hollow tetrahedron. The red edges have a sharpness of 0, 1, 2, 3, and 10 respectively, and subdivision has been applied 5 times.

Figure 11: Part of a screenshot of *Geri's Game*, a short movie by Pixar.

6.1. Beyond Catmull-Clark

In the 20 years following the invention of the original subdivision methods a number of new schemes have been developed. A large number of these new schemes are in the spirit of Catmull-Clark. A review of these schemes is available, in which the authors focus on three major themes [Cas11]:

- Theory for analysing subdivision surfaces,

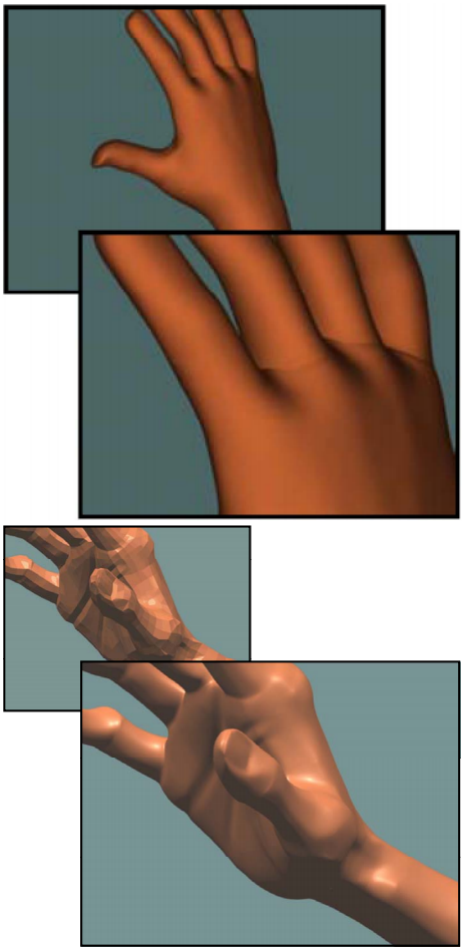


Figure 12: A comparison between the old NURBS techniques (Woody's hand, top [*Toy Story*]) and the methods covered in the paper (Geri's hand, bottom [*Geri's Game*])

- tools for integrating subdivision schemes into existing applications, and
- the development of surfaces with improved properties such as fairness, compatibility, or the removal of artefacts.

The two latter themes have practical applications, yet only the second theme about integrating existing schemes has found widespread use. This raises the question about why subdivision surfaces are popular in areas where they are already accepted and are little used in ares where they are not.

Despite the large amount of research done on improving the subdivision schemes they almost always appear in their original, unmodified-forms even though these schemes have some unavoidable curvature problems. The author lists several reasons:

- The amount of work specifically for Catmull-Clark creates an ecosystem together with large amounts of optimised code which can be difficult to change.
- Catmull-Clark is the de facto standard of how subdivision systems should behave.
- Catmull-Clark is good enough for its purpose. The unavoidable curvature problems rarely appear in practice.
- The current subdivision methods are simple and general. The alternatives may be considerably more complex or have limited applicability.

6.2. OpenSubdiv

Catmull-Clark is currently still often used as it produces one of the best results although it is computationally expensive. This is why it is mostly used in animation where render farms can be used to perform these computations.

Because Catmull-Clark subdivision is so computationally expensive Pixar in collaboration with Microsoft initiated a project called *OpenSubdiv* which aims to be faster, more efficient, and more flexible than the standard subdivision. In order to achieve a speedup the computations are moved to the GPU instead of the CPU since GPUs can perform these calculations at a much higher rate. Note that OpenSubdiv is not an application or a tool but an API which can be integrated into existing software. An example of this is the 3D graphics application Maya which integrates the OpenSubdiv API so that fast subdivision can be used [Pix18].

In OpenSubdiv it is possible to create sharp edges in a similar manner as we described in the previous sections. In OpenSubdiv you can tag edges and edges chains as creases while providing a sharpness value ranging from 0-10, with a sharpness value ≥ 10 treated as infinitely sharp [Pix18].

6.3. NURBS and subdivison

Currently both NURBS and subdivision are both still actively used both with different use cases. NURBS is mainly used in Computer Aided Design (CAD) and subdivision is popular in the field of animation. Both representation are built on uniform B-splines but the extend this foundation in different ways. However both methods have useful features which could be used in the other field. Subdivision surfaces for example can be of arbitrary degree and non-uniform which would be good additions to current subdivision surfaces. Subdivision surfaces on the other hand can be of arbitrary topology which would be useful in CAD. Currently there is some research on how to create NURBS-compatible subdivision surfaces as can be seen in Figure 13 [Cas10].

7. Summary

The paper [DKT98] provides an elegant solution to the semi-sharp crease problem which subdivision surfaces have. The implementation is easy and the results are satisfactory as can be seen in the results section. From 1998 and onward the

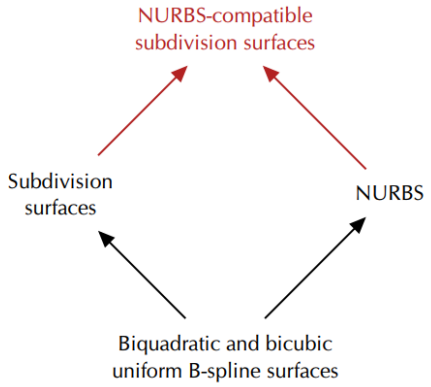


Figure 13: The current incompatibility between NURBS and existing subdivision surfaces. The arrows represent subset relations [Cas10].

technique has often been used by large production companies like Pixar. In recent years Pixar helped to create an open source standard for subdivision called OpenSubdiv which is actively used by other large companies.

NURBS is also still actively used, mainly in CAD programs. Some successful research has been done on how to create NURBS compatible subdivision surfaces [Cas10] but we could not find an active use of this method.

References

[Cas10] CASHMAN T. J.: *NURBS-compatible subdivision surfaces*. PhD thesis, University of Cambridge, 2010. 1, 2, 7, 8

[Cas11] CASHMAN T. J.: Beyond Catmull-Clark? A survey of advances in subdivision surface methods. *Computer Graphics Forum 31*, 1 (2011), 42–61. doi:10.1111/j.1467-8659.2011.02083.x. 6

[CC78] CATMULL E., CLARK J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–355. doi:10.1016/0010-4485(78)90110-0. 3

[DKT98] DEROSE T., KASS M., TRUONG T.: Subdivision surfaces in character animation. *Proceedings of the 25th annual conference on computer graphics and interactive techniques - SIGGRAPH 98* (1998). doi:10.1145/280814.280826. 1, 2, 3, 4, 7

[HFN*14] HUANG Y.-C., FENG J.-Q., NIESSNER M., CUI Y.-M., YANG B.: Feature-adaptive rendering of Loop subdivision surfaces on modern GPUs. *Journal of Computer Science and Technology 29*, 6 (2014), 1014–1025. doi:10.1007/s11390-014-1486-x. 4, 5

[Loo87] LOOP C.: *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, January 1987. URL: https://www.microsoft.com/en-us/research/publication/smooth-subdivision-surfaces-based-on-triangles/. 2

[Pix18] PIXAR: Introduction - OpenSubdiv, Jul 2018. URL: http://graphics.pixar.com/opensubdiv/docs/intro.html. 7

[ZP00] ZORIN D., PETER S.: *Subdivision for modeling and animation: SIGGRAPH 2000 course notes*. 2000. 2, 3