Dynamic Refinement of Deformable Triangle Meshes for Rendering

Kolja Kähler Jörg Haber Hans-Peter Seidel

Computer Graphics Group
Max-Planck-Institut für Infomatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
E-mail: {kaehler, haberj, hpseidel}@mpi-sb.mpg.de

Abstract

We present a method to adaptively refine an irregular triangle mesh as it deforms in real-time. The method increases surface smoothness in regions of high deformation by splitting triangles in a fashion similar to one or two steps of Loop subdivision. The refinement is computed for an arbitrary triangle mesh and the subdivided triangles are simply passed to the rendering engine, leaving the mesh itself unchanged. The algorithm can thus be easily plugged into existing systems to enhance visual appearance of animated meshes. The refinement step has very low computational overhead and is easy to implement. We demonstrate the use of the algorithm in our physics-based facial animation system.

1. Introduction

In real-time computer animation, polygon meshes are a popular surface representation due to the high throughput on current hardware. The drawback is that a polygonal surface is piecewise planar, and we thus have to find a balance between high visual quality (more polygons) and high frame rates (less polygons).

A polygon model of a static surface can be built such that a compromise between the represented amount of detail and computational complexity is achieved. However, when the surface is animated by moving the mesh nodes, the resulting mesh doesn't adapt well to the deformation. Bends and folds can only appear at the edges of the mesh and are thus limited by its initial connectivity. This leads to the unfortunate situation that the model has to provide enough polygons to accommodate for possible deformations, even though the surface can be represented well enough with a far lower number of polygons in its undeformed state. A model should thus be dynamically refined at runtime where required.

In this paper, we present a technique to render adaptively refined versions of a triangle mesh. Refinements are computed only in those areas where the mesh is deformed. Since these refinements are used for rendering only and can be easily computed on the fly, we do *not* update the original triangle mesh. This is advantageous for two reasons: first, the application doesn't have to deal with dynamic mesh connectivity, making integration into existing systems a simple plug-in operation. Second, the refined triangles are not retained between rendered frames, so additional memory usage is kept to a minimum.

Our method has been integrated into a framework for physics-based animation of polygonal models, see Figure 1. A spring mesh is created from the initial triangle mesh: vertices correspond to point masses and edges correspond to springs. The spring mesh deforms in the simulation loop and its nodes are used to update the triangle mesh. During animation it shows that the resolution of the triangle mesh is often too coarse in highly deformed regions. However, the simulation can be decoupled from the rendering: without changing the resolution of the spring mesh (which defines the precision of the simulation), we can render a smoother version of the deformed triangle mesh.

2. Related Work

A large number of surface representations are used in computer animation. Among the most popular choices are polynomial patches, polygons, and subdivision surfaces.

Polynomial patches have long been used in modeling and animation [13, 4]. Surfaces built from such patches are defined by a relatively coarse control mesh. Animation of the surface can be achieved by deforming this mesh. The generated surface is inherently smooth, but for complex geometry built from multiple patches, preservation of smoothness conditions across patch boundaries becomes difficult. Patches can be refined globally via knot insertion [2] or locally using hierarchical methods [5]. For real-time rendering, patches are usually tessellated using uniform [14, 9]

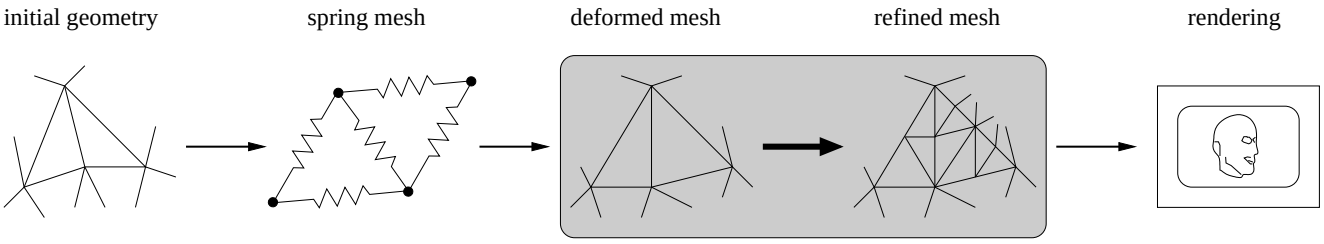

Figure 1. Overview of our physics-based animation system. Without affecting the internal state of the simulation or the triangle mesh itself, refinements are computed from the deformed mesh and passed to the rendering engine.

or adaptive schemes [12] to exploit fast polygon rendering hardware. As the surface deforms, the vertices of the tessellation have to be recomputed. For refined surfaces, the computations become more complicated and expensive.

Polygonal models are popular due to their simplicity, flexibility, and the availability of efficient graphics hardware. Adaptive refinement of arbitrary triangles meshes is a recent topic in multi-resolution editing [7, 8]. These methods are very powerful, but the underlying machinery is complex and currently not applicable in real-time environments.

In physics-based animation, spring meshes are typically composed of quadrilaterals or triangles, and the mass points of the spring mesh are identified with the vertices of the rendered surface [10, 18]. Adaptive refinement of such a mass-spring system is non-trivial [6]. Volino *et al.* [19] propose an efficient method to smooth polygonal geometry, which is applied to deformations caused by a mass-spring simulation. Their method interpolates interior points of arbitrary polygons, given its vertices and vertex normals. Using a regular subdivision of the initial geometry, smooth surfaces can be generated on the fly for rendering, similar to our approach. The tessellation does not adapt to surface curvature, though. In the context of facial animation, Seo *et al.* [16] describe the application of level-of-detail techniques, generating not only coarser geometry, but also coarser animation control for far away viewpoints.

Subdivision surfaces [11, 15] bridge the gap between spline patches and polygon meshes in many respects, combining the easy handling of meshes with the well-defined properties of a parametric surface. Defined over an initial quadrilateral or triangle control mesh, an arbitrarily close approximation to a smooth limit surface can be generated by repeatedly refining the mesh using simple rules. The limit surface can either interpolate or approximate the control mesh nodes, depending on the subdivision rules. subdivision surfaces are also suitable for use in computer animation [3]. A very regular mesh of subdivision connectivity is required, which often makes an initial remeshing step necessary when an irregular mesh is given. In general, the refinement operator is applied uniformly to a subdivision surface. Zorin *et al.* [21] describe a method ("adaptive synthesis") that selectively computes refined triangles by temporarily creating the needed parent triangles.

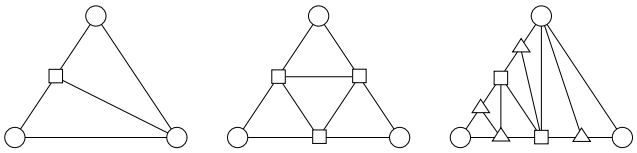

Figure 2. Examples of split configurations. From left to right: One single edge split, three edges split once, two edges split twice.

3. Method Overview

A generic adaptive refinement algorithm employing some surface curvature criterion can be stated recursively:

```
refine(region r):
  c := curvature(r)
  if (c > threshold)
    subdivide(r, c)
    for all sub-regions s in r
      refine(s)
  else
    draw(r)
```

Even if the tail-recursion is flattened by transformation into a loop, two cost factors remain: the curvature has to be evaluated multiple times on the initial region (albeit on smaller and smaller parts), and the changes caused by a subdivision step have to be stored in the geometry before the subregions can be examined (or temporary storage must be allocated per sub-region on each level of recursion.) In our approach, we minimize these costs by evaluating the curvature only once: based on the outcome, we perform up to two

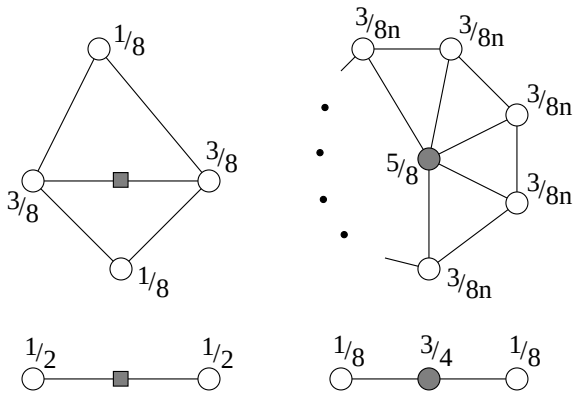

Figure 3. Left: Computing the position of a first-level split vertex (■). Right: Displacing a vertex in the input mesh (●). The top row shows the weights for an interior vertex, the bottom row shows the boundary case.

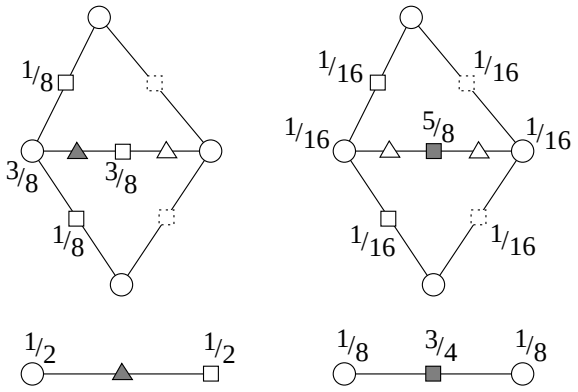

Figure 4. Left: Computing the position of a second-level split vertex (▲) from original mesh vertices (○) and first-level split vertices (□). Right: Displacing a first-level split vertex (■). Non-existent first-level vertices (stippled □) are approximated by linear interpolation.

refinements in one step, thus eliminating the need for storing the altered geometry for further evaluation. It is directly drawn and discarded:

```
refine(region r):
  c := curvature(r)
  if (c > threshold)
    region s = subdivide1or2(r, c)
    draw(s)
  else
    draw(r)
```

In our implementation, the refinement procedure is applied to the triangle mesh just before rendering, as shown in Figure 1. We make use of a number of adjacency relations that are defined on a triangle mesh, such as circulating through the vertices adjacent to a given vertex, finding the triangles sharing a given edge etc. On this behalf, we use a data structure based on half-edges as described by Campagna *et al.* [1].

Each edge of the given deformed mesh is examined to decide whether it should be split into two or more parts, causing subdivision of the adjacent triangles. Using the new degrees of freedom provided by the split vertices, we compute a smoother re-triangulation approximating the input mesh. For the smoothing, simple local rules are used that borrow from the subdivision idea. We don't generate any new vertices in the interior of an original triangle, thus avoiding evaluation of new interior edges and keeping the number of possible new triangulations manageable. The re-triangulation is efficiently created by a table lookup operation. The resulting triangle set is then rendered instead of the original triangle, unsplit triangles are rendered as usual. Figure 2 shows examples of split configurations.

The computed refinements are not reflected in the input mesh, they are computed dynamically for each frame and discarded after rendering. Thus, undoing refinements is not necessary, and the input mesh remains unaltered. We also do not retain any information about splits between frames.

4. The Algorithm

Our method creates sub-triangles by splitting triangle edges once into two or twice into four parts. We start by iterating over all edges of the input mesh, deciding whether to split them once or twice. Since the splitting of each edge is carried out in one single pass, there is no recursion involved. In a second pass, the re-triangulation of each triangle is obtained from the split configuration along its edges.

4.1. Splitting Criterion

We assume that the quality of the triangulation of the undeformed mesh is good enough for the intended application. Therefore we only want to split an edge if the curvature of the surrounding mesh region has increased during mesh deformation. As a simple and efficient test, we use the dot product between the vertex normals at both ends of an edge. If this scalar value drops below the value that has been precomputed for the undeformed geometry, there is more "bending" and the edge is marked for splitting once or twice, depending on the difference of the dot products.

This criterion only uses the vertex normals of the existing nodes in the mesh. More complex criteria can be used as well, e.g. measuring discrete curvature on the mesh [17].

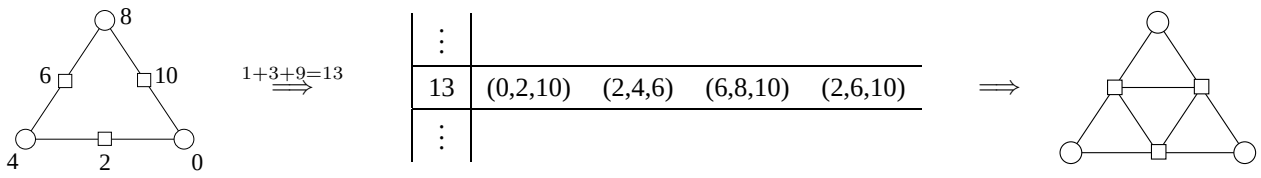

Figure 5. A triangle is split and re-triangulated using a lookup table.

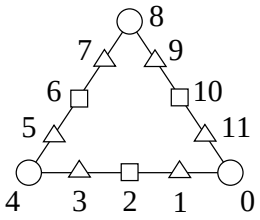

Figure 6. Numbering of original (○) and split vertices (□, △), starting at the first vertex of edge 0 in a triangle.

The vertex normal dot product has proven to be sensitive to the kind of deformations that occur in our application. Additionally, it has the advantage of extremely low evaluation cost, provided that vertex normals have been computed before.

4.2. Vertex Smoothing

After having determined the split configuration for each triangle, the vertex positions of the resulting sub-triangles are computed. We want to make sure that this operation

- is computationally cheap;
- only changes the surface locally;
- results in a close approximation of the input mesh;
- handles mesh boundaries correctly.

Our approach to selective refinement is inspired by Loop subdivision [11] in the variant proposed by Warren [20]. The Loop subdivision scheme also applies mid-edge splitting, and computing the refined surface only requires quick averaging of old and new vertices with their immediate neighbors. We'd like to point out that our method doesn't produce surfaces with any particular degree of continuity, but just a smoother-looking approximation of the original.

Vertex positions corresponding to the first (i.e. mid-edge) split of the triangle edges are calculated similarly to the Loop scheme by weighted averaging. Figure 3 shows the vertices that take part in these computations and the associated weights. Since we don't want to change the input mesh, the new positions for the original vertices are temporarily buffered.

For triangle edges that have been split twice, the vertex positions corresponding to the second-level splits are obtained in a similar fashion, see Figure 4. Here, we have to use the previously computed positions of the first-level split vertices. Since generally not all edges of a triangle are split, some of these vertices may not have been computed before. In this case, we simply take the mid-point of the respective original edge.

Furthermore, the original mesh vertices are not smoothed again for second-level splits, contrary to the proper Loop subdivision scheme. In this way, we avoid complicated updates involving adjacent triangles and keep these vertices closer to their original locations.

4.3. Generating Sub-Triangles

Once the new vertex positions have been computed, sub-triangles are created by connecting these points and then passed to the rendering engine. To speed up the re-triangulation step, we use a lookup table that has one entry per split configuration. Each entry contains a sequence of vertex indices, which represents a valid tessellation of the original triangle. The points in each triangle are indexed according to Figure 6. If $s_i \in \{0, 1, 2\}$ denotes the number of splits that have been applied to edge $i \in \{0, 1, 2\}$ of the current triangle, the index into the table is computed from the ternary digits s_i as $s_0 + 3s_1 + 9s_2$, yielding 27 possible combinations. Figure 5 illustrates the table lookup mechanism.

Each input triangle can be split into a maximum of ten sub-triangles. No cracks appear in the generated mesh, since adjacent triangles have a common edge and thus share the split configuration along this edge, see Figure 7.

4.4. Time Coherent Splitting

For proper shading and texturing, vertex normal and texture coordinates of a new vertex are interpolated linearly from the neighboring vertices along the edge. The neighboring vertices are either original mesh vertices or previously created split vertices. Due to the nature of intensity value interpolation in Gouraud shading, the re-triangulation

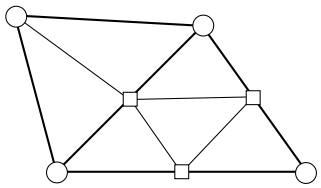

Figure 7. Adjacent triangles are split consistently.

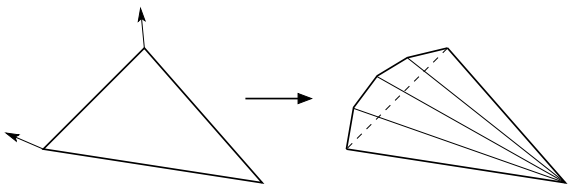

Figure 8. Thin triangles that are generated during adaptive refinement are aligned to the direction of minimum curvature.

pattern in the lookup table does not affect the rendered output. For flat shading, however, triangle normals have to be computed for the generated sub-triangles. If the triangulation is chosen only on the basis of the triangle's current split configuration, shading artifacts may appear: a split that is introduced from one frame to the next may lead to a completely different triangulation, causing abrupt changes in the surface normals of the sub-triangles. This can be alleviated by taking the history of refinements on a triangle into account. If an edge is split further than in the previous frame, a re-triangulation is generated that is equivalent to refining that previous triangulation pattern.

Though we can't avoid maintaining some sort of history, we still don't have to store split triangles between frames: it is feasible to enumerate all *sequences* of splits that can be applied to a triangle, from the zero-length sequence containing no splits at all to the sequences of length six containing two splits on each edge in every possible order. We can construct a table of 271 entries, where each line corresponds to one of these sequences. Each table entry is automatically constructed by an algorithm that splits and subdivides a triangle following the corresponding sequence.

In this way, if the history of applied splits is stored along with the mesh, one can generate time-coherent re-triangulations. However, splits can only be taken back in reverse order, otherwise artifacts may again appear. Additional overhead is induced by the more complicated maintainance of data structures and the bigger lookup table (deteriorated data locality). In practice we usually avoid the overhead of time coherent splitting, since flat shading is rarely used in our applications.

5. Results

The current implementation of our refinement scheme delivers good results in our facial animation environment. Figure 9 shows a detail of the deformed mouth region during animation. The rendered mesh is significantly smoothed in interior and boundary regions, reducing shading artifacts and improving the silhouette of the opened mouth.

In our experiments, there was no noticeable difference in frame rate when running with or without dynamic refinement. This was to be expected in the simulation context, where the computational load is mainly caused by the evaluation of the physics model and not by the rendering stage.

We found that the explicit implementation of one to two refinement levels paid off, because there is no overhead for recursion and maintaining dynamic data structures. This will of course only hold under the assumption of a sufficiently tessellated undeformed mesh. Though the method could be extended to more than two splits per edge, our experiments have shown that this level of refinement is sufficient for the moderate deformations our initial model experiences.

When looking at the refined triangle meshes as shown in Figure 9, one clearly notices many long and thin triangles. Usually, this is an indication of a badly generated triangle mesh. Here, however, the thin triangles are exactly what we want. Figure 8 shows that the automatically generated sub-triangles are aligned to the direction of minimum curvature, thus mimicking the alignment of folds on real skin.

6. Future Work

We would like to extend our method in several ways. Visual quality can be further improved by refining the mesh along silhouette edges. To achieve this, appropriate splitting criteria have to be developed. Also, it would be interesting to investigate the effects of other smoothing schemes, since we currently don't interpolate, but only approximate the input surface. If the input geometry has been produced from a finer mesh, one could do even better than smoothing: detail information can be stored locally and used to place generated split vertices on the surface, as has been exercised in multi-resolution editing.

Finally, the rendering performance can be improved by generating re-triangulations that can be encoded as triangle strips and / or triangle fans.

References

[1] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed Edges — a Scalable Representation for Triangle Meshes. *Journal*

of Graphics Tools, 3(4):1–11, 1998.

[2] C. de Boor. *A Practical Guide to Splines*. Springer–Verlag, New York, 1978.

[3] T. DeRose, M. Kass, and T. Truong. Subdivision Surfaces in Character Animation. In *Computer Graphics (SIGGRAPH '98 Conf. Proc.)*, pages 85–94, July 1998.

[4] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, San Diego, CA, 1993.

[5] D. R. Forsey and R. H. Bartels. Hierarchical B-Spline Refinement. In *Computer Graphics (SIGGRAPH '88 Conf. Proc.)*, volume 22, pages 205–212, Aug. 1988.

[6] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive Refinement for Mass-Spring Simulation. In *7th Eurographics Workshop on Animation and Simulation*, pages 31–45, 1996.

[7] L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution Shape Deformations for Meshes with Dynamic Vertex Connectivity. *Computer Graphics Forum*, 19(3):249–260, 2000.

[8] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *Computer Graphics (SIGGRAPH '98 Conf. Proc.)*, pages 105–114, July 1998.

[9] S. Kumar, D. Manocha, and A. Lastra. Interactive Display of Large-Scale NURBS Models. In *Symposium on Interactive 3D Graphics 1995*, pages 51–58, Apr. 1995.

[10] Y. Lee, D. Terzopoulos, and K. Waters. Realistic Modeling for Facial Animations. In *Computer Graphics (SIGGRAPH '95 Conf. Proc.)*, pages 55–62, Aug. 1995.

[11] C. T. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[12] J. W. Peterson. Tessellation of NURB Surfaces. In *Graphics Gems IV*, pages 286–320. Academic Press, 1994.

[13] L. Piegl and W. Tiller. *The NURBS Book*. Springer–Verlag, New York, 2. edition, 1997.

[14] A. Rockwood, K. Heaton, and T. Davis. Real-Time Rendering of Trimmed Surfaces. In *Computer Graphics (SIGGRAPH '89 Conf. Proc.)*, volume 23, pages 107–116, July 1989.

[15] J. E. Schweitzer. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, 1996.

[16] H. Seo and N. Magnenat-Thalmann. LoD Management on Animating Face Models. In *Proc. IEEE Virtual Reality 2000*, pages 161–168, 2000.

[17] G. Taubin. Estimating the Tensor of Curvature of a Surface from a Polyhedral. In *Proc. International Conference on Computer Vision*, pages 902–907, 1995.

[18] A. Van Gelder. Approximate Simulation of Elastic Membranes by Triangulated Spring Meshes. *Journal of Graphics Tools*, 3(2):21–41, 1998.

[19] P. Volino and N. Magnenat-Thalmann. The SPHERIGON: A Simple Polygon Patch for Smoothing Quickly your Polygonal Meshes. In *Proc. Computer Animation '98*, pages 72–79, 1998.

[20] J. Warren. Subdivision Methods For Geometric Design. Unpublished manuscript. Preprint available at `http://www.cs.rice.edu/jwarren/papers/book.ps.gz`.

[21] D. Zorin, P. Schröder, and W. Sweldens. Interactive Multiresolution Mesh Editing. In *Computer Graphics (SIGGRAPH '97 Conf. Proc.)*, pages 259–268, Aug. 1997.

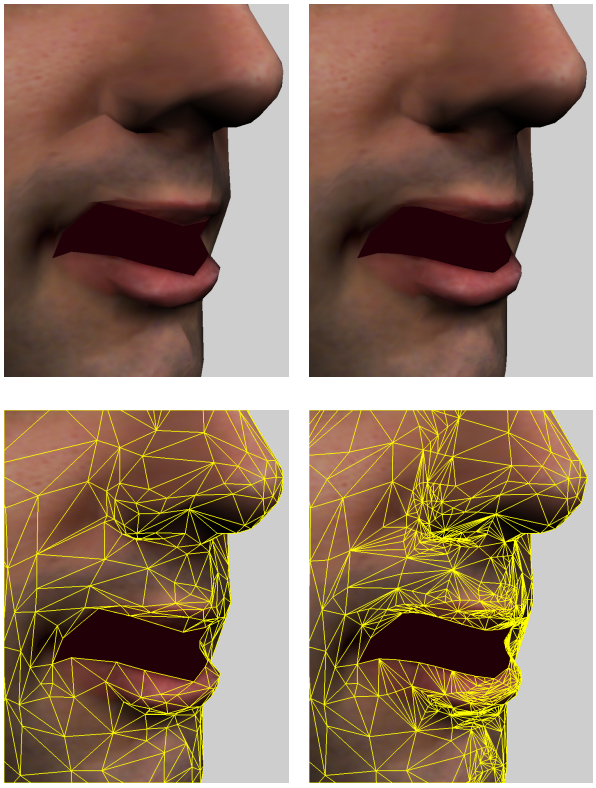

Figure 9. Left: Initial face mesh (3247 triangles). Right: Dynamically refined mesh (4767 triangles). Teeth and tongue have been removed for clarity. The animation runs at approx. 5 fps on an `sgi` O2 with 250Mhz in both cases.