GPU Supported Patch-based Tessellation for Dual Subdivision

Fengtao Fan[*]
University of Kentucky

Fuhua (Frank) Cheng[†]
University of Kentucky

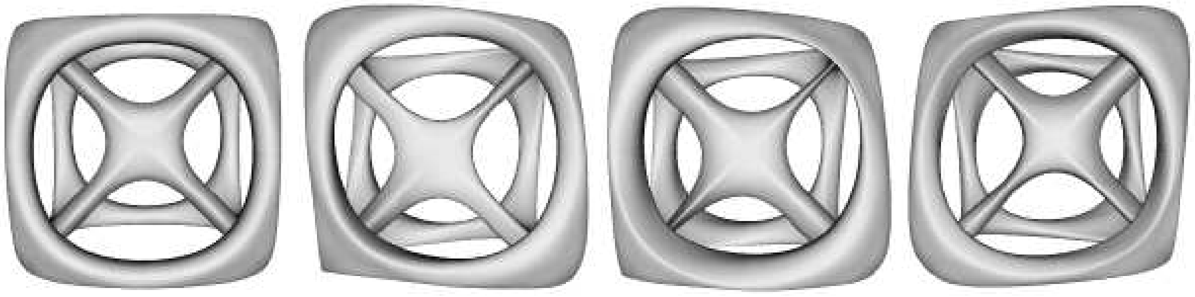

Figure 1: *Full evaluation (depth 5) of a toy model (66 patches) at 56fps on NVIDIA GeForce 8800 GT*

Abstract

A novel patch-based tessellation method for a dual subdivision scheme, the Doo-Sabin subdivision, is presented. Patch-based refinement for face-split subdivision schemes such as Catmull-Clark subdivision or Loop subdivision has been widely studied. But there is no patch-based tessellation algorithm for dual subdivision scheme [Shiue et al. 2005] yet. The method presented in this paper is the first attempt to fill up that gap. The new method uses an 1D array to hold vertices; it creates a patch corresponding to a vertex in the original mesh and does not have any numerical roundoff gaps on patch boundaries. These characteristics are different from those of patch-based refinements for face-split subdivision schemes. Experimental results show that our algorithm achieves real time tessellation performance for moderate meshes.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Curve, Surface, Solid, and Object Representations; I.3.8 [Computing Methodologies]: Computer Graphics—Graphics Data Structures and Data Type

Keywords: subdivision surfaces, programmable graphics hardware, patch-based refinement

1 Introduction

Subdivision surfaces are gaining popularity in several areas, such as geometric modeling and computer animation. One of the reasons is its capability in representing objects of arbitrary topology with only one surface. A subdivision surface is generated by recursively refining a given mesh (called control mesh) until a limit surface is reached. Hence a subdivision surface is determined by the control mesh and the refining rule (or, subdivision scheme). Most of the better known stationary subdivision schemes can be classified into two categories [Zorin and Schröder 2000]: *face-split* and *vertex-split*. The first category contains Loop [Loop 1987], modified butterfly [Zorin et al. 1996] for triangular meshes, and Catmull-Clark [Catmull and Clark 1978] and Kobbelt [Kobbelt 1996] for quadrilateral meshes. Doo-Sabin [Doo and Sabin 1978], midedge and biquartic subdivision schemes are in the vertex-split category. There are several other subdivision schemes not in these two categories such as the $\sqrt{3}$ subdivision [Kobbelt 2000]. Implementation of this subdivision is very simple due to its repetitive structure.

With the recent dramatic advancement of their computation power, *Graphics Processing Units* (GPUs) are no longer limited to rendering purpose only, they have been used for other purposes as well. Actually there is an effort to develop GPUs into *General Purpose GPUs* (GPGPU) [Luebke et al. 2004]. CUDA (*Compute Unified Device Architecture*) developed by NVIDIA is a very typical platform for general purpose computation of GPUs. To utilize the computation power of GPUs, several algorithms have been implemented on GPU to achieve higher performance, such as marching cubes, particle systems [Kipfer et al. 2004; Kolb et al. 2004] and collision detection [Govindaraju et al. 2003]. More examples can be found in the CUDA Zone of NVIDIA's website or in [Nguyen 2007]. In particular, there is a big collection of literature about efficient tessellation of subdivision surfaces on GPU. A fast rendering of Loop subdivision surface by dividing the control mesh into different patches with triangle pairs and their intermediate neighboring vertices is proposed in [Pulli and Segal 1996]. Employing the forward differencing technique for hardware supported implementation of Loop subdivision surface is explored in [Bischoff et al. 2000]. A general meshing scheme method for adaptive subdivision surface rendering which sends groups of vertices to the graphics pipeline is presented in [Bóo et al. 2001]. A parallel evaluation of subdivision surfaces on graphics hardware is presented in [Padrón et al. 2002]. Patch-based adaptive tessellation for Catmull-Clark surfaces with displacement mapping is shown in [Bunnell 2005]. [Shiue et al. 2003] proposes a generic framework for tessellating subdivision surfaces on GPU. All these algorithms fall into the category of patch-based method. Patch-based refinement has the advantages of locality, efficiency for recursive refinement, and adaptivity et al. Another type of tessellation and rendering of subdivision surfaces is to use lookup tables for pre-computed data, like the table driven tessellation algorithm in [Bolz and Schröder] and a general subdivision kernel based on spiral-enumerated fragment meshes in [Shiue et al. 2005]. There are other applications dealing

[*]e-mail: ffan2@uky.edu
[†]e-mail: cheng@cs.uky.edu

with subdivision surface on GPU, e.g. deformation of subdivision surfaces on GPU [Zhou et al. 2007].

Most of existing tessellation methods for subdivision surfaces can not be directly applied to dual subdivision schemes, such as Doo-Sabin subdivision or mid-edge subdivision except the general kernel [Shiue et al. 2005]. Especially, to the best of our knowledge, there is no patch-based tessellation algorithm for vertex-split subdivision surfaces [Shiue and Peters 2005] [Shiue et al. 2005] although patch-based tessellation techniques for face-split subdivision surfaces have been widely studied. In this paper, We try to fill up this gap by introducing a patch-based tessellation algorithm for Doo-Sabin subdivision surface. Our patch-based tessellation for Doo-Sabin subdivision surface differs from those for face-split subdivision surfaces in several aspects. First, a patch in our algorithm corresponds to a vertex of the initial control mesh. In general, the number of vertices of a mesh is smaller than the number of faces. Hence, there are less patches in our method. Second, our method uses a 1D array to store patch vertices while those for face-split subdivision employ a 2D array. Third, there is no numerical roundoff gaps between patch boundaries in our method, while such gaps exist in those patch-based methods for face-split method and need to be taken care of. The reason that the third aspect is true is because in our method, two adjacent Doo-Sabin vertex patches share a common strip of faces. A new common strip of faces is formed by new vertices generated for faces of the current common strip during the subdivision process. Since these new vertices depend on faces of the current strip only, therefore, even though both adjacent patches need to compute these new vertices themselves, they would get exactly the same points and, consequently, the same new common strip of faces. Hence, there is no roundoff gaps in our case at all. Further more, note that the general subdivision kernel in [Shiue et al. 2005] is also suitable for Doo-Sabin subdivision surfaces. But this approach needs to subdivide the initial mesh twice on CPU while ours needs only one subdivision as pre-processing. Several experimental results show that our patch-based tessellation for Doo-Sabin subdivision achieves realtime performance for moderate meshes. For instance, our algorithm has a 35fps rate for full evaluation of a toy model to depth 5 (Figure 1).

The remaining part of the paper is arranged as follows. Details of the new method are presented in Section 2. Performance of the new method and test results are shown in Section 3. Concluding remarks and future work are given in Section 4.

2 Patch-based tessellation and implementation

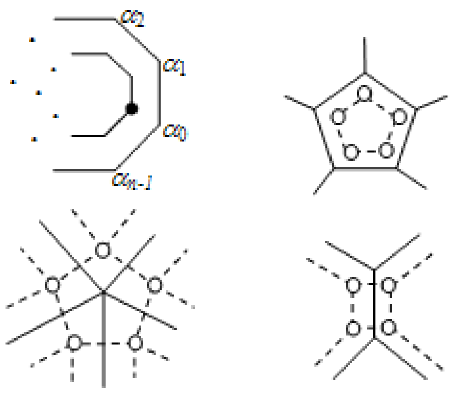

Figure 2: *Upper left: stencil for Doo-Sabin subdivision,* $\alpha_0 = (n+5)/4n$ *and* $\alpha_i = (3 + 2\cos((2i\pi)/n))/4n$ *(*$0 < i < n$*); Upper right: f-face for a face; Lower left: v-face for a vertex; Lower right: e-face for an edge.*

A dual subdivision scheme refines the current mesh by (conceptually) splitting its vertices according to their valences to generate new vertices, and then connecting the new vertices to form new edges and new faces. The refined mesh contains three types of faces: *v-faces*,*e-faces* and *f-faces*, as they correspond to vertices, edges, and faces in the current mesh, respectively. By repeatedly performing this refinement process on a given control mesh, one gets a limit surface in the end. Such a limit surface is called a subdivision surface of the dual subdivision scheme. Doo-Sabin subdivision scheme is a typical dual scheme. The refinement stencil for Doo-Sabin subdivision and the corresponding *v-face*,*e-face* and *f-face* are shown in Figure2. Note that if one can generate each patch of the limit surface independently by refining an appropriate subset of the control mesh (called *patch-based tessellation*), then one can parallelize the limit surface generation process by generating all the patches of the limit surface simultaneously on a GPU or some special hardware. Our intention here is to develop patch-based tessellation techniques for Doo-Sabin subdivision surfaces.

Developing *patch-based tessellation*) techniques for *face-split* subdivision schemes is relatively easy. This is because for each face-split subdivision scheme, such as Catmull-Clark or Loop, each patch of the limit surface corresponds to a face in the control mesh. Therefore, it is straightforward to identify the subset of the control mesh that has to be tessellated to generate a specific patch of the limit surface. This is not true for a dual subdivision scheme. Patches of the limit surface of a dual subdivision scheme do not correspond to faces of the initial control mesh. Instead, they correspond to vertices of the initial control mesh (see Figure 3). We call such a patch *vertex patch*.

Given an input mesh $\mathbf{M}$ with N_v vertices, we first subdivide it once to generate a *v-face* for each vertex. The refined mesh is then broken into N_v submeshes. Each submesh, consisting of a *v-face* and its adjacent faces, corresponds to a vertex patch. With the establishment of an initial submesh for each vertex patch, Doo-Sabin subdivision is then applied to each submesh independently. The final refined mesh of $\mathbf{M}$ is the union of the final refined meshes of all these submeshes. Note that two adjacent vertex patches share a strip of quadrilateral faces in the refined mesh of every level. In the initial submesh of a vertex patch, the *v-face* and adjacent *f-faces* (generated for faces of $\mathbf{M}$ adjacent to

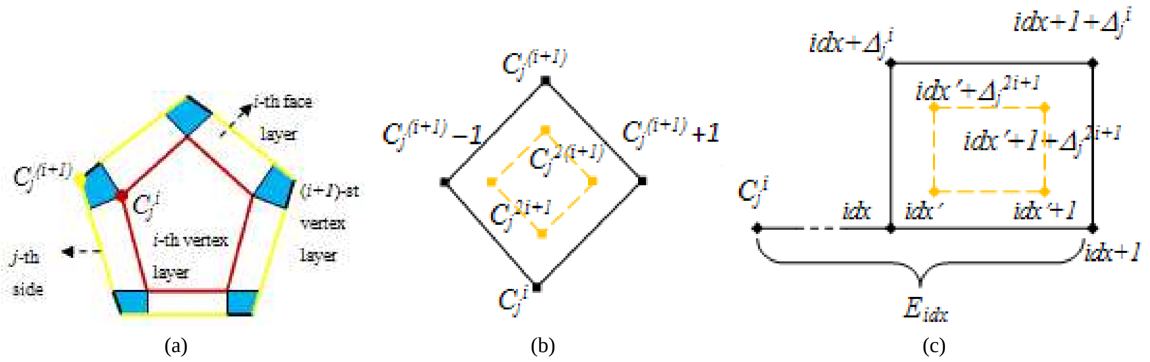

Figure 4: *(a) face layouts and vertex layouts; (b) indices of a corner quad and its subdivided new corner quad; (c) indices of a regular quad and its subdivided new regular quad.*

this vertex) could be arbitrary polygons; but the adjacent *e-faces* are all quadrilateral faces. If the input mesh is a quadrilateral mesh, all faces adjacent to a *v-face* are quadrilaterals (see the Upper left case of Figure 3). A submesh of this type is called a *semi-quad* submesh. In the following, we first show how to subdivide these semi-quad submeshes using Doo-Sabin subdivision scheme. We then extend our patch-based tessellation algorithm to arbitrary meshes by introducing a special operation for arbitrary *f-faces*.

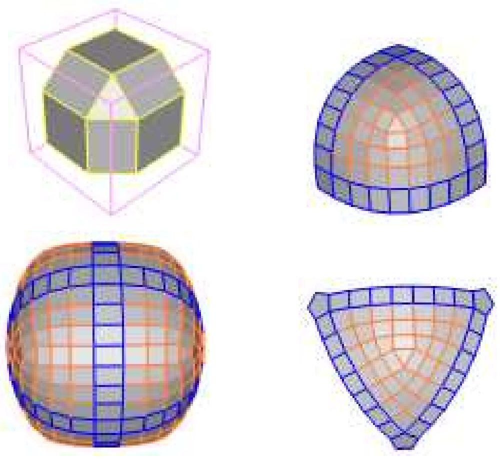

Figure 3: *Upper left: an initial submesh of a vertex patch in a cube mesh. Upper right: the refined submesh of this patch after* 3 *levels of subdivision; blue quadrilaterals are shared with other patches; Lower left: strips of blue quadrilaterals are shared between patches; Lower right: a none semi-quad vertex patch at subdivision depth* 3 *whose generator vertex has a valence of* 3*; note that the layout is similar to that of the patch in upper right.*

2.1 Subdivision of *Semi-quad* Vertex Patches

The layout of a semi-quad vertex patch is completely determined by the valence n of the corresponding vertex in the original mesh, called a *generator vertex*. A valence of n indicates that the *v-face* in the initial submesh has n vertices. In this subsection, the *generator vertex* of a semi-quad vertex patch is assumed to has a valence of n.

At each refinement level of the *semi-quad* vertex patch, all faces can be assigned into different *face layers* naturally by their positions. The inner most *v-face* is considered in the 0-th layer. In general, quadrilaterals adjacent to i-th layer faces are assigned to the $(i+1)$-st layer. Similarly, we can classify vertices into different *vertex layers*. Vertices of the inner most *v-face* are in the 0-th vertex layer. Vertices shared by quadrilaterals in the i-th and $(i+1)$-st layers are assigned into the i-th vertex layer, as in Figure 4(a). In each face layer except the 0-th layer, there are n quadrilaterals corresponding to the *f-faces* of the initial semi-quad submesh, called *corner quads*. Other quadrilaterals are considered as *regular quads*. We also divide the vertices in each vertex layer into two categories: *corner vertex* and *regular vertex*. Vertices of the single face in the 0-th face layer are all *corner vertices*. The two vertices on the diagonal of a *corner quad*, which are from two consecutive face layers, are two *corner vertices* on their own layers, respectively. An illustration is given in Figure 4(a). Thus there are exactly n corner vertices in each vertex layer. It is natural to divide the vertices in each layer into n sides. The i-th side contains the vertices from the i-th corner vertex to the $(i+1)$-st corner vertex (excluding $(i+1)$-st corner vertex itself).

There are several useful observations on the number of vertices in the layout of a vertex patch. At subdivision level $d \geq 1$,

1. there are $2^{d-1}+1$ vertex layers. The layer index l is assumed to be $0 \leq l \leq 2^{d-1}$.
2. In vertex layer l, there are $2l+1$ vertices on each side, thus $(2l+1)n$ vertices in all.
3. The total vertices in the refined submesh of a vertex patch at level d is $(2^{d-1}+1)^2 n$.

We simply assign all vertices of a vertex patch into an array in sequential order by assigning the inner layers first as shown in Figure 6. The vertex indices for the l-th vertex layer is from $l^2 n + 1$ to $(l+1)^2 n$. The j-th $(0 \leq j \leq n-1)$ corner vertex in the l-th layer is in the position of $C_j^l = l^2 n + 1 + j(2l+1)$. A vertex on the j-th side of the l-th vertex layer is connected to a regular vertex on the same side of the $(l+1)$-st vertex layer to form a bounding edge of a regular quad. The index difference between this pair of connected vertices in consecutive layers is $\Delta_j^l = (2l+1)n + 2j + 1$. Then vertex indices of a corner quad are C_j^l, $C_j^{l+1}+1$,C_j^{l+1} and $C_j^{l+1}-1$ as shown in Figure 4(b). A regular quad contains vertices with indices idx, $idx+1$, $idx+1+\Delta_j^l$ and $idx+\Delta_j^l$ as shown in Figure 4(c), where idx is a vertex on the j-th side of the l-th vertex layer. Therefore, at subdivision level d, we can extract all faces from the sequence of vertices by constructing quadrilaterals of the above forms. More precisely,

1. The inner most face is obtained by connecting vertices 1 to n;

2. For a vertex idx between 1 and $(2^{d-1})^2 n$, first determine its vertex layer l and its side j in this layer. Extract a regular quad with vertices idx, $idx+1$, $idx+1+\Delta_j^l$ and $idx+\Delta_j^l$; if vertex idx is a corner vertex, extract a corner quad with vertices C_j^l, $C_j^{l+1}+1$,C_j^{l+1} and $C_j^{l+1}-1$.

Here all the additions and subtractions are considered in the modulo sense, i.e., $C_0^l - 1 = C_{n-1}^l + 2l$ and so on.

For Doo-Sabin subdivision, new vertices are generated for each face. So far, we can extract all faces from a sequential array for subdivision. The question then is, how to store the new vertices back into the sequential array. At subdivision level $d+1$, there are 2^d+1 face layers. Note that the quadrilaterals in the $(i+1)$-st face layer of level d creates the quadrilaterals in the $2(i+1)$-st face layer in the refined submesh. Consequently, vertices on the new quadrilaterals are in the $(2i+1)$-st and $(2i+2)$-nd vertex layers of the refined submesh. Precisely, each corner quad in the $(i+1)-st$ face layer with vertices C_j^i, $C_j^{i+1}+1$,C_j^{i+1} and $C_j^{i+1}-1$ is mapped to the corresponding corner quad in the $(2i+2)$-nd layer which has vertices C_j^{2i+1}, $C_j^{2i+2}+1$, C_j^{2i+2} and $C_j^{2i+2}-1$, as shown in Figure 4(b). For each regular quad on the j-th side, we need a new parameter E_{idx} to determine the indices of the new vertices. Let E_{idx} is just the number of edges between $idx+1$ and C_j^{i+1}. We have $E_{idx} = idx + 1 - C_j^{i+1}$. Then the new regular quad has the vertices of $idx' = C_j^{2i+1} + 2E_{idx}$, $idx'+1$, $idx' + \Delta_j^{2i+1}$ and $idx' + \Delta_j^{2i+1}$, as in Figure 4(c). The above analysis shows the process of subdividing a semi-quad vertex patch. We next extend this tessellation method to arbitrary meshes.

2.2 Generalization

For an arbitrary mesh, *f-faces* in the initial submesh of a vertex patch no longer have to be quadrilaterals. If they have more than 4 vertices, they can not fit into the sequential array discussed above any more. The layout of the refined submesh of a vertex patch depends on the valence n of its *generator vertex* and how many vertices on each *f-face*. In this subsection, we use the assumption that, for the given vertex patch, the valence of the generator vertex is n and the numbers of vertices of the n *f-faces* are $m_1, m_2, \ldots, m_n$, respectively. Note that, although the initial submesh of an arbitrary vertex patch looks quite different from that of a semi-quad vertex patch, their refined submeshes after a few times of subdivision look very much alike. In fact, the only difference between these submeshes are just the n corner quads in the outer most face layer if the valences of their generator vertices are both n. This observation leads us to a minor modification of the tessellation algorithm for semi-quad vertex patch to handle arbitrary vertex patches.

Let M_v be the maximum of vertex valences of the original mesh and let M_f be the maximum of mesh faces' vertex numbers of the original mesh. We append $M_v \times M_f$ spaces to the $(2^{d-1}+1)^2 n$ sequential spaces for a vertex patch at subdivision level d such that the m_j vertices of the j-th *f-face* are stored in the slots from $j \times M_f$ to $(j+1) \times M_f$. Note that the corner quads and corner vertices in the outer most face layer and vertex layer are no longer valid now. But we still keep the spaces for them. These corner quads and corner vertices now act as *flags*. To differentiate them from others, they are called *virtual corner quads* and *virtual corner vertices*, respectively (see Figure 6).

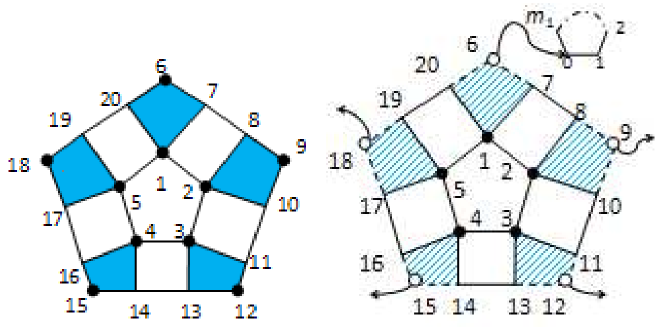

Figure 6: *Left: a semi-quad vertex patch at subdivision depth* 1*, where solid black circles represent corner vertices and blue quadrilaterals are corner quads; Right: black circles represent virtual corner vertices and slashed quadrilaterals are virtual corner quads.*

Now we can extract faces for an arbitrary vertex patch as we did for a semi-quad vertex patch except for those virtual corner quads. When the considered face is the j-th virtual corner quad, we retrieve the m_j vertices of the *f-face* from the appended spaces. Once new vertices have been generated for a virtual corner quad in the new subdivision process, new vertices are stored back into the identical positions in the appended spaces. Note that each virtual corner quad has three vertices shared with adjacent regular quads. Using the above approach, these three vertices will be stored twice, once for the adjacent regular quads and once for the virtual corner quad. This additional storage is necessary to maintain a retrieving mechanism that works for both semi-quad vertex patches and arbitrary vertex patches.

In summary, we have the following algorithm for subdividing a mesh from depth d to $d+1$.

For a vertex with index $idx \in [1, (2^{d-1})^2 n]$ do

- Identify the layer l and the side j it belongs to;
- If the vertex idx is a corner vertex, extract vertices of the corner quad Q_c: $C_j^l, C_j^{l+1}+1, C_j^{l+1}$ and $C_j^{l+1}-1$.
  - If Q_c is a virtual corner quad,
    - ○ Extract vertices of the j-th *f-face* from the appended $M_v \times M_f$ spaces;
    - ○ Subdivide this face by Doo-Sabin subdivision scheme;
    - ○ Write new vertices to identical positions of a new array; at the same time, write the three none virtual corner vertices to their corresponding new positions;
  - Otherwise,
    - ○ Subdivide this corner quad by Doo-Sabin subdivision scheme;
    - ○ Write vertices of the new corner quad to a new array;
- Extract vertices of the regular quad with indices Q_r of $idx, idx+1, idx+1+\Delta_j^l$ and $idx+\Delta_j^l$;
  - ○ Compute the step size for the new indices: $E_{idx} = idx + 1 - C_j^l$;
  - ○ Subdivide this regular quad by Doo-Sabin subdivision scheme;

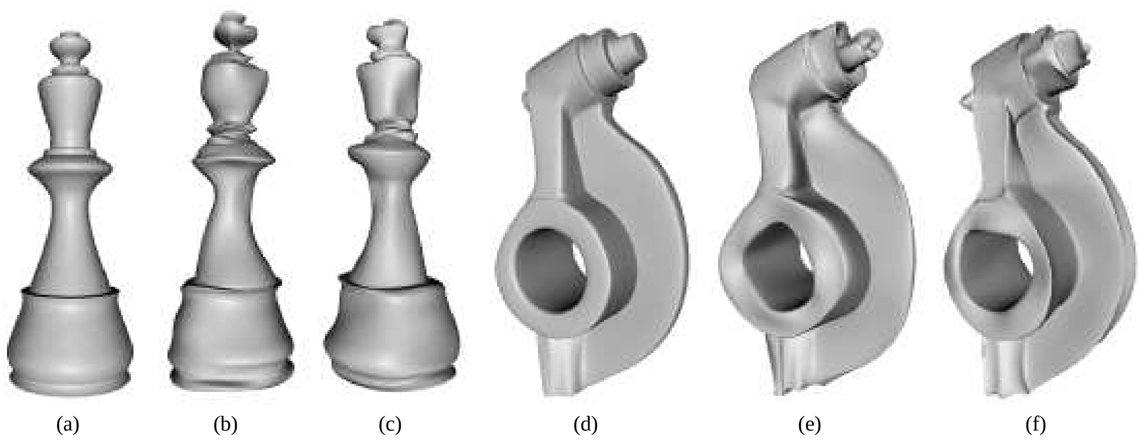

Figure 5: *(a)-(c):full evaluation of a chessman model (*314 *patches) to depth* 5 *at* 14*fps; (d)-(f):full evaluation of the rocker arm model (*354 *patches) to depth* 5 *at* 19*fps; (a) and (d) are the original subdivision surfaces; (b), (c), (e) and (f) are deformed surfaces.*

- ○ Write vertices of the new regular quad to a new array;

3 Performance and results

After breaking a given mesh into vertex patches, the patches can be evaluated independently. This intrinsic parallel characteristic makes it highly suitable for running on GPU. Our patch-based tessellation algorithm uses a 1D array to represent a vertex patch. The input mesh is first subdivided once on CPU to initialize the 1D array representations for the resulting vertex patches. After initialization, the evaluation is confined to each 1D array itself. At the expected subdivision depth, the connectivity information needed for rendering can easily be retrieved from these sequences of vertices.

Our implementation is performed on a desktop with a 3.0GHz Pentium 4 CPU, 512M RAM and a GeForce 8800 GT GPU (512M). We use the CUDA platform for GPU programming. To avoid rendering via the CPU, we use vertex buffers for vertices of the input meshes and their normals. By using the OpenGL Interoperability of CUDA, we can process these vertex buffers in GPU computation. After processing, it is directly rendered on GPU. The rendering process is highly accelerated by using vertex buffers. In order to measure the performance of our program, we wobble the vertices of the input mesh along their normals and reevaluate the mesh to the prescribed depth, i.e. 5 in our examples. Figure 1 shows reevaluating a toy model with 66 patches to depth 5 at 56fps. The ant model in Figure 7 with 298 patches is reevaluated to depth 5 at 14fps. More examples are shown in Figure 8 on a helix model with 505 patches, Figure 5(a) - Figure 5(c) on a chessman model with 314 patches and Figure 5(d)-Figure 5(f) on a rocker arm model with 354 patches. All these examples show that our patch-based tessellation algorithm achieves near realtime performance. Compared to our implementation on CPU, the GPU implementation runs about 20 times faster. For instance, the performance for the rocker arm model on CPU is less than 1fps.

4 Conclusion and Future Work

In this paper, a patch-based tessellation algorithm for Doo-Sabin subdivision scheme is developed. Our patch-based tessellation creates a vertex patch for every vertex in the input mesh. This vertex patch is represented by a 1D array. All connectivity information can be easily retrieved from the 1D sequence. All vertex patches are evaluated independently. There is no numerical roundoff gaps issue because the shared parts between patches are strips of quadrilaterals. With the intrinsic parallelism of our patch-based tessellation, it is easily adapted on GPU. Our GPU implementation achieves near realtime performance for moderate meshes. In general, patch-based refinements possess the flexibility of adaptive refinement [Bunnell 2005]. One of our future works is to investigate adaptive tessellation of our patch-based method.

Acknowledgements

The ant model is obtained from the repository of the Princeton Shape Benchmark.

References

BISCHOFF, S., KOBBELT, L. P., AND SEIDEL, H.-P. 2000. Towards hardware implementation of loop subdivision. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM, New York, NY, USA, 41–50.

BOLZ, J., AND SCHRÖDER, P. Evaluation of subdivision surfaces on programmable graphics hardware. http://www.multires.caltech.edu/pubs/gpusubd.pdf.

BÓO, M., AMOR, M., DOGGETT, M., HIRCHE, J., AND STRASSER, W. 2001. Hardware support for adaptive subdivision surface rendering. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM, New York, NY, USA, 33–40.

BUNNELL, M., 2005. Gpu gems 2: Programming techniques for high-performance graphics and general-purpose computation (gpu gems).

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 350–355.

DOO, D., AND SABIN, M. 1978. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design 10*, 356–360.

GOVINDARAJU, N. K., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*,

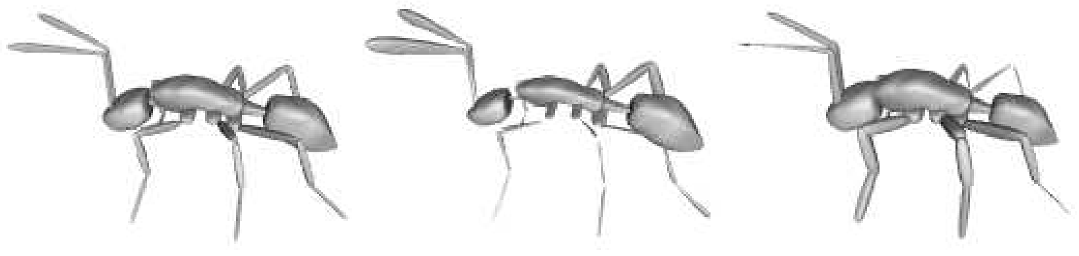
(a) (b) (c)

Figure 7: *Full evaluation of an ant model(*298 *patches) to depth* 5 *at* 14*fps; (a) the original subdivision surface; (b) and (c) deformed surfaces.*

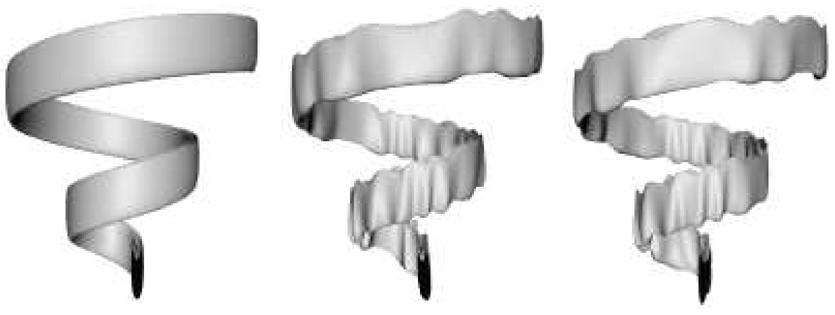
(a) (b) (c)

Figure 8: *Full evaluation of a helix model(*505 *patches) to depth* 5 *at* 14*fps; (a) the original subdivision surfaces; (b) and (c) deformed surfaces.*

Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 25–32.

KIPFER, P., SEGAL, M., AND WESTERMANN, R. 2004. Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, New York, NY, USA, 115–122.

KOBBELT, L. 1996. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Computer Graphics Forum*, 409–420.

KOBBELT, L. 2000. $\sqrt{3}$-subdivision. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 103–112.

KOLB, A., LATTA, L., AND REZK-SALAMA, C. 2004. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, New York, NY, USA, 123–131.

LOOP, C. 1987. *Smooth subdivision surfaces based on triangles*. Master's thesis, Utah University.

LUEBKE, D., HARRIS, M., KRÜGER, J., PURCELL, T., GOVINDARAJU, N., BUCK, I., WOOLLEY, C., AND LEFOHN, A. 2004. Gpgpu: general purpose computation on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, ACM, New York, NY, USA, 33.

NGUYEN, H. 2007. *Gpu gems 3*. Addison-Wesley Professional.

PADRÓN, E. J., AMOR, M., BÓO, M., AND DOALLO, R. 2002. Efficient parallel implementations for surface subdivision. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 113–121.

PULLI, K., AND SEGAL, M. 1996. Fast rendering of subdivision surfaces. In *Proceedings of the eurographics workshop on Rendering techniques '96*, Springer-Verlag, London, UK, 61–70.

SHIUE, L.-J., AND PETERS, J. 2005. A pattern-based data structure for manipulating meshes with regular regions. In *GI '05: Proceedings of Graphics Interface 2005*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 153–160.

SHIUE, L.-J., GOEL, V., AND PETERS, J. 2003. Mesh mutation in programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 15–24.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime gpu subdivision kernel. *ACM Trans. Graph. 24*, 3, 1010–1015.

ZHOU, K., HUANG, X., XU, W., GUO, B., AND SHUM, H.-Y. 2007. Direct manipulation of subdivision surfaces on gpus. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 91.

ZORIN, D., AND SCHRÖDER, P., 2000. Subdivision for modeling and animation. ACM SIGGRAPH 2000 Course Notes.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1996. Interpolating subdivision for meshes with arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 189–192.