Realtime Loop Subdivision on the GPU

UNIVERSITY OF FLORIDA

Minho Kim and Jörg Peters

http://www.cise.ufl.edu/research/SurfLab

subdivision surfaces

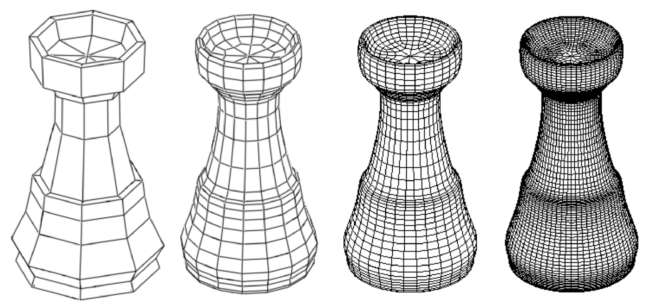


Figure 1. Catmull-Clark subdivision.

Figure 2. Loop subdivision.

challenge 1: mesh fragments

Re-representation of the control net as overlapping mesh fragments yields per-patch parallelism and simplifies shaders.
Loop subdivision requires different mesh fragments than Catmull-Clark subdivision.

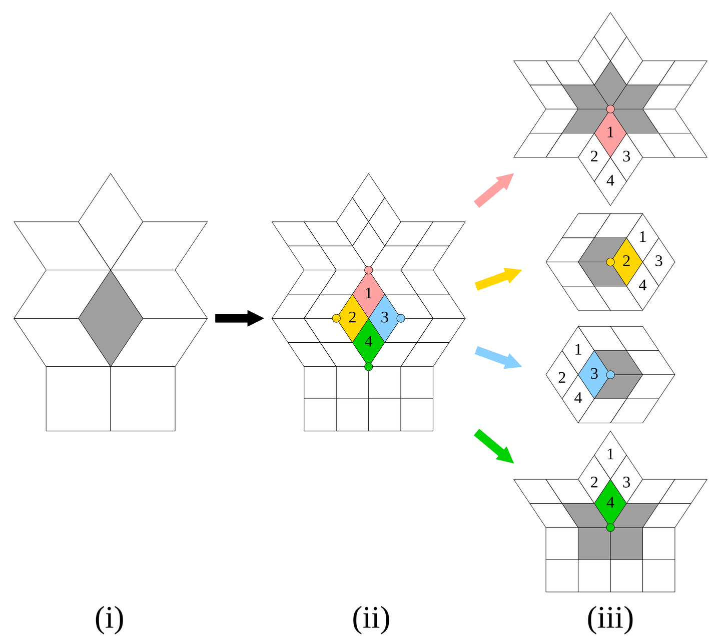


Figure 3. Catmull-Clark subdivision.

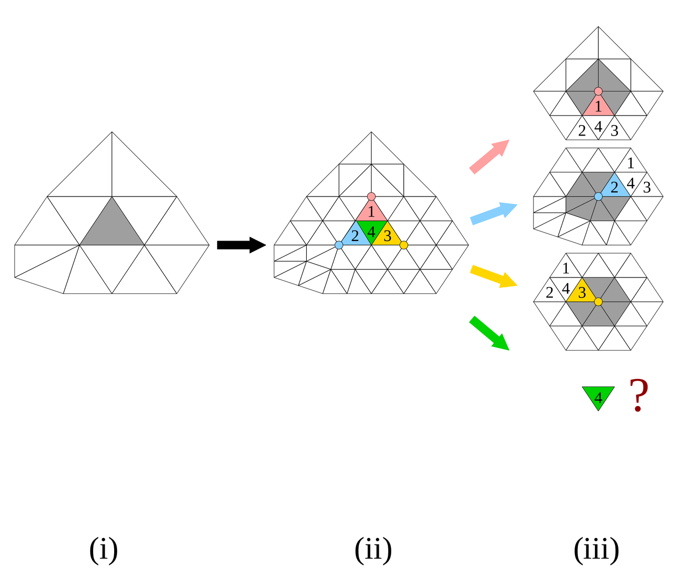


Figure 4. Loop subdivision.

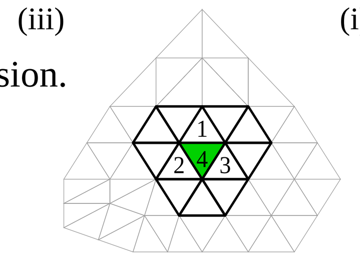


Figure 5. Control mesh for the isolated center facet patch.

challenge 2: indexing

Parallel stream processing requires mapping of a local control net to a 1D array. Fortunately, a consistent indexing over different LODs exists.

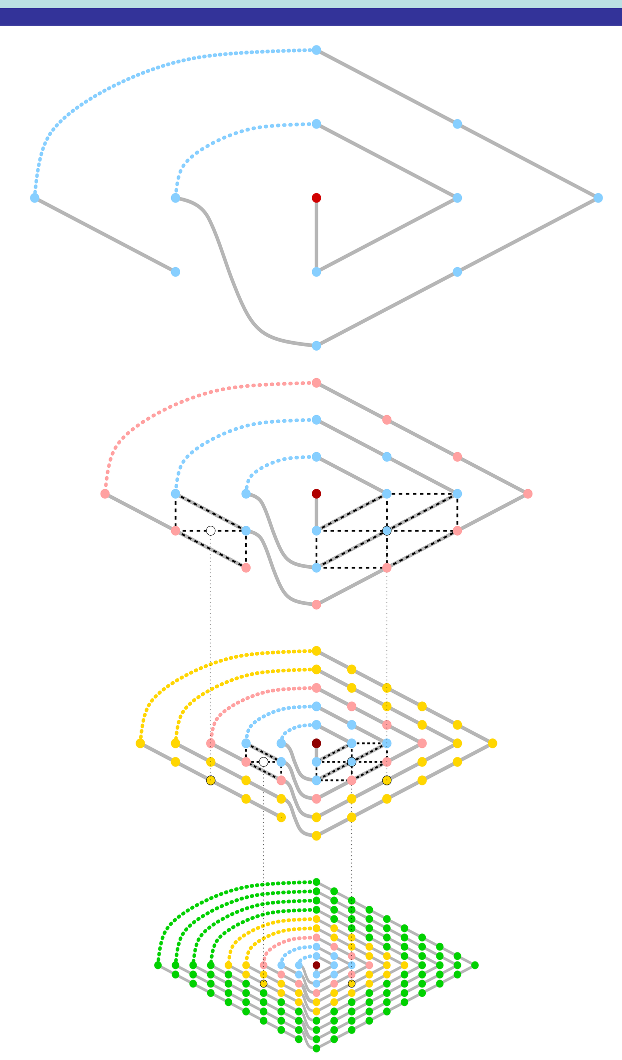


Figure 6. Vertex patch indexing.

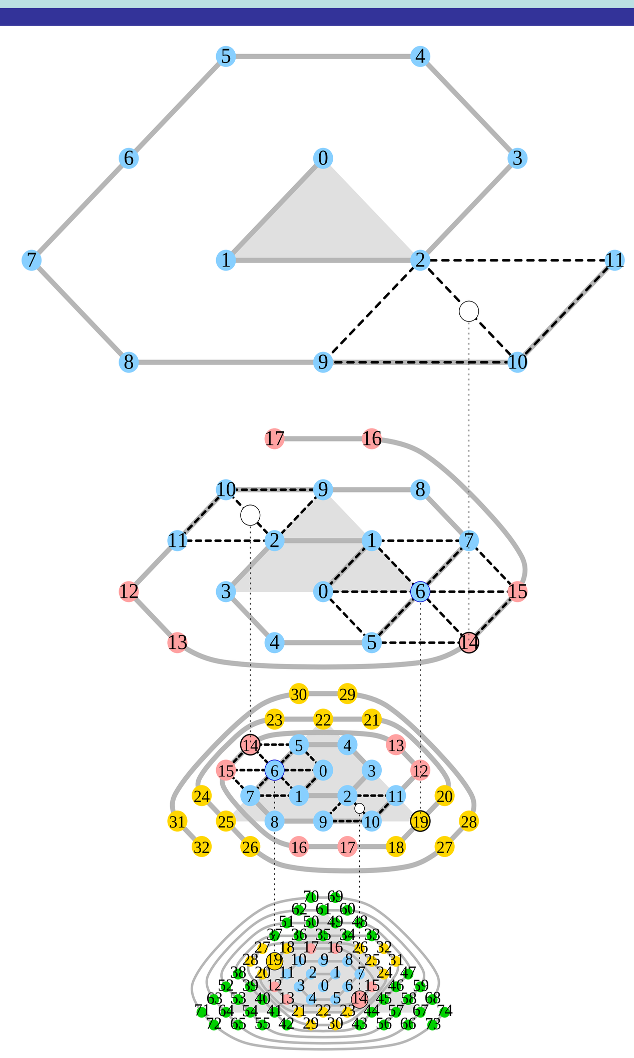


Figure 7. Facet patch indexing.

In ***A realtime GPU subdivision kernel*** (SIGGRAPH 2005), Shiue et al. showed that, in principle, all major features of subdivision algorithms can be realized in the framework of highly parallel stream processing.

Shiue et al. tested the approach by implementing Catmull-Clark subdivision, with semi-smooth creases and global boundaries, in programmable graphics hardware, at near realtime speed.

Here, we report on the challenges when adapting the approach to Loop subdivision.

challenge 3: workflow

Pattern tables facilitate consistent subdivision of mesh fragments on the GPU.

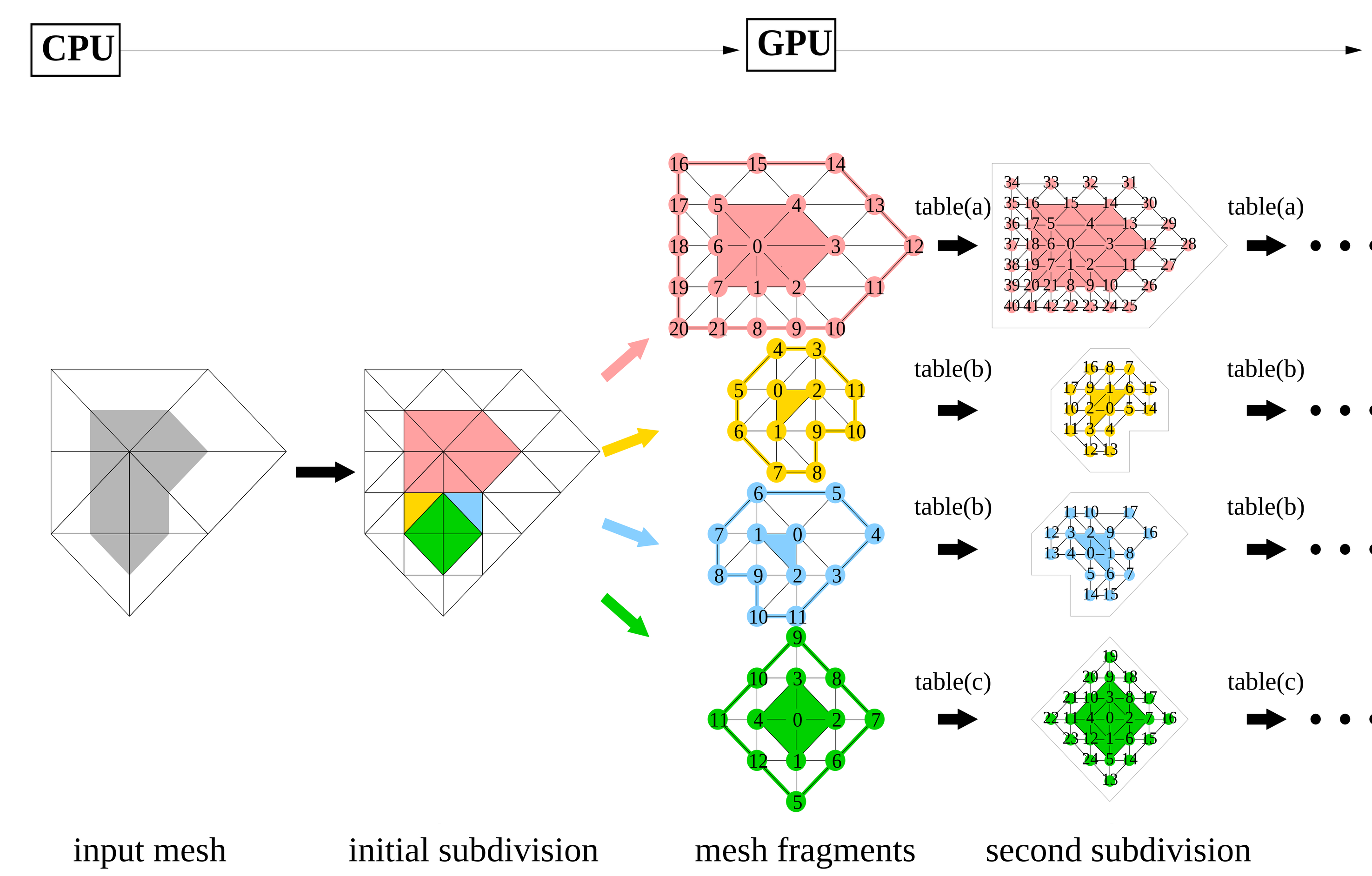


Figure 8. Workflow. Subdivision uses the pattern tables in Figure 9.

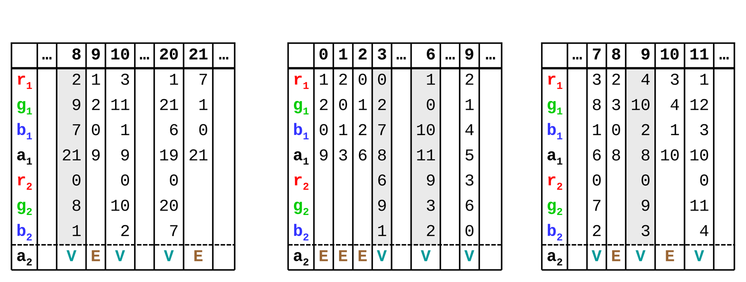


Figure 9. Pattern tables for (a) valence 7 vertex patch (b) facet patch (c) valence 4 vertex patch.

```
a2 == V:
p=((((r1+g1)+(b1+a1))+(r2+g2))+10*b2)/16;

a2 == E:
p=((3*(r1+g1)+(b1+a1))/8;
```

Figure 10. Logic of the shader program for consistent control point generation.

related work

[1] David Brickhill, ***Practical Implementation Techniques for Multiresolution Subdivision Surfaces***, Proceeding of Game Developers Conference 2001.

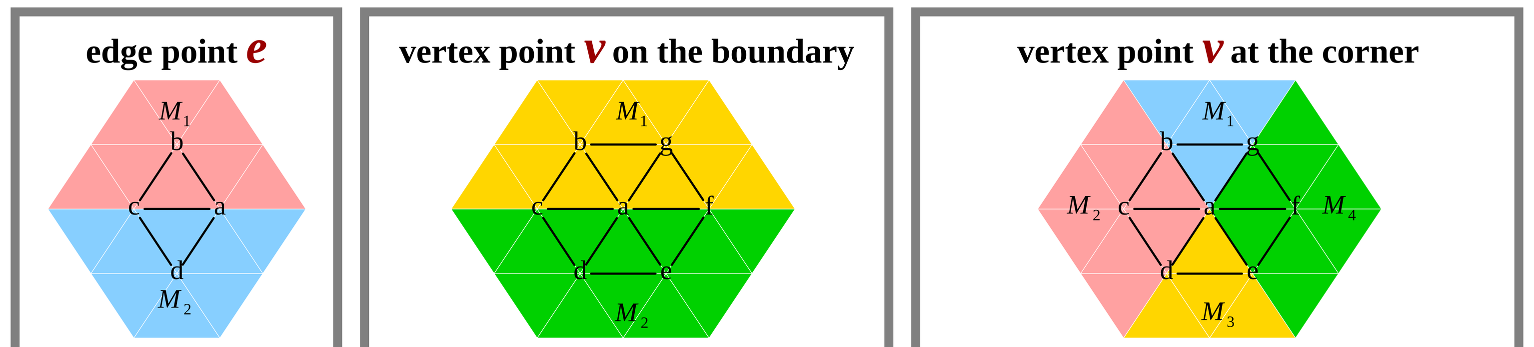
[2] Jeffrey Bolz and Peter Schröder, ***Rapid Evaluation of Catmull-Clark Subdivision Surfaces***, Web3D 2002 Symposium.

[3] Jeff Bolz and Peter Schröder, ***Evaluation of Subdivision Surfaces on Programmable Graphics Hardware***, submitted for publication, 2003.

[4] Michael Guthe, Ákos Balázs and Reinhard Klein, ***GPU-Based Trimming and Tessellation of NURBS and T-Spline Surfaces***, SIGGRAPH 2005.

challenge 4: watertight boundaries

By observing symmetries, identical points can be generated consistently on different mesh fragments.



Edge point:

$$e(M_1)=\frac{1}{8}(3(c+a)+(b+d))$$
$$e(M_2)=\frac{1}{8}(3(a+c)+(d+b))$$

Vertex point on the boundary:

$$v(M_1)=\frac{1}{16}((((b+g)+(d+e))+(c+f))+10a)$$
$$v(M_2)=\frac{1}{16}((((e+d)+(g+b))+(f+c))+10a)$$

Vertex point at the corner:

$$v(M_1)=v(M_2)=\frac{1}{16}((((b+g)+(d+e))+(c+f))+10a)$$
$$v(M_3)=v(M_4)=\frac{1}{16}((((e+d)+(g+b))+(f+c))+10a)$$

Figure 11. Symmetric evaluation for watertight boundaries between mesh fragments.

example of consistent computation

$$8 \leftarrow \frac{1}{16}((((2+9)+(7+21))+(0+8))+10\times 1)$$
$$3 \leftarrow \frac{1}{16}((((0+2)+(7+8))+(6+9))+10\times 1)$$
$$6 \leftarrow \frac{1}{16}((((1+0)+(10+11))+(9+3))+10\times 2)$$
$$9 \leftarrow \frac{1}{16}((((4+10)+(2+8))+(0+9))+10\times 3)$$

Figure 12. Consistent computation of the same boundary point in each of four refined mesh fragments with the notation and colors of Figure 8.

performance

fps	liver		stomach		mechpart		venus	
config. \ depth	4	5	4	5	4	5	4	5
1	9.70	6.27	8.53	4.41	7.62	4.60	4.27	1.88
2	22.83	15.63	19.42	11.43	18.28	13.07	9.70	5.24
3	18.32	9.15	12.08	4.85	13.62	6.81	5.04	1.90
4	13.91	8.42	11.43	5.71	10.33	6.04	5.42	2.40

Table 1. Benchmark results (frames per second).

	CPU	GPU	system memory	video memory	# of shaders per patch[1]	off-screen rendering buffer	double buffering	buffer size	data round-trip removal
1	Pentium 4 (2.40GHz)	ATi Radeon 9700 Pro (Omega driver 2.5.97a)	1GB	128MB (AGP 4x)	2	pbuffer	no	2048x1024	none
2	Pentium 4 (3.00GHz)	nVidia GeForce 6800GT (driver 71.84)	1GB	256MB (AGP 8x)	1	pbuffer	yes	2048x256	PBO/VBO
3	Pentium M (1.60GHz)	nVidia Geforce 6200 (driver 70.87)	512MB	128MB[2] (PCI Express 16x)	1	pbuffer	yes	2048x256	PBO/VBO
4	Pentium 4 (2.80GHz)	ATi Radeon X800 (Omega driver 2.5.97a)	1GB	256MB (AGP 8x)	1	pbuffer	no	2048x1024	none

1: For some GPUs, each shader is divided into two due to the limitation of shader length.
2: 32MB is dedicated for GPU and 96MB is shared with main memory.

Table 2. Benchmark configurations. Note the importance of round-trip removal.

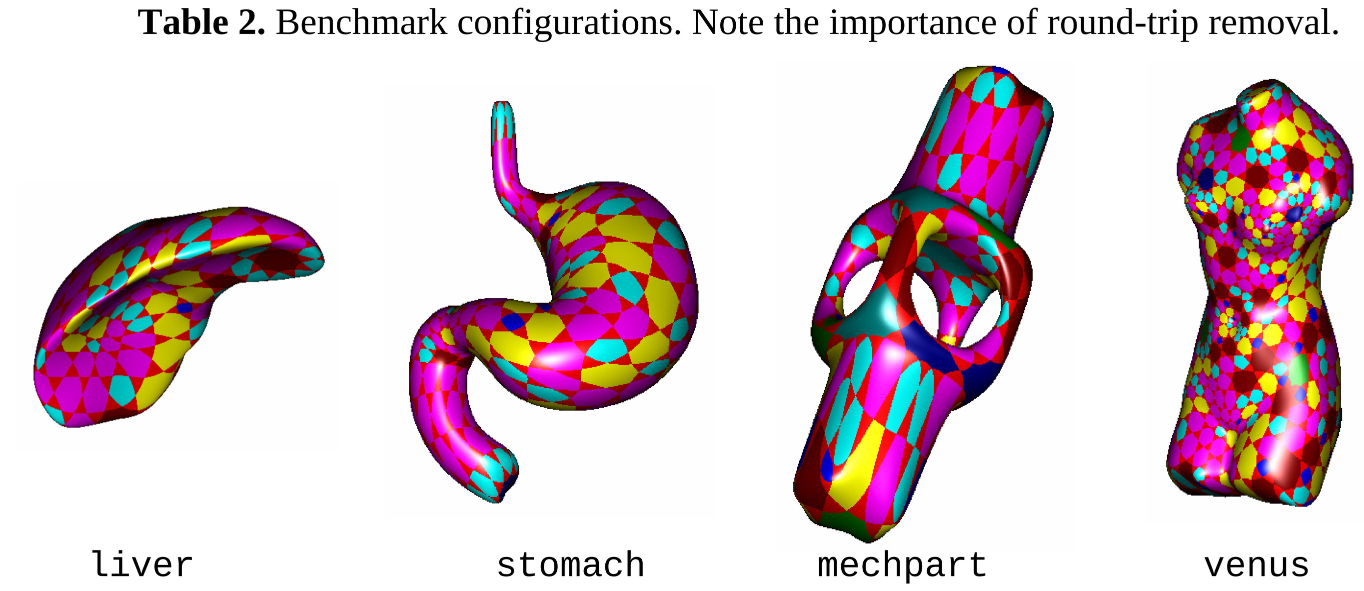


Figure 13. Test subdivision surfaces. (# of facets/#of vertices)