Subdivision Meshes in GPU

Young-Jun Kim

KAIST (Korea Advanced Institute of Science and Technology)
Contents

- Introduction
- Background
  - Subdivision meshes
  - GPU
- Related Works
- Problem and Idea
- Conclusion
Introduction

- The subdivision meshes are developed for representing the characters and objects having smooth shape for the animations and games
- Subdivision meshes in the movies
  - Geri’s Game (Pixar 1997)
  - A Bug’s Life (Pixar 1998)
  - Meet The Robinsons (Disney 2007)
Subdivision Meshes (1)

- Recursively refine a polygonal mesh

- Number of iteration determines Level-Of-Detail (LOD)
- Provide infinite LOD

Original control mesh → Smoother surface by recursive processing
Subdivision Meshes (2)

- Two phase process
  - Refinement phase: creates new vertices and reconnects to create new triangles
  - Smoothing phase: computes new positions for the vertices
Advantage of Subdivision (1)

- **Efficiency**
  - Modeling is easy

- **Arbitrary topology**
  - Classic spline approaches have great difficulty with control meshes of arbitrary topology.

Standard valence : 4 (regular point ○ )
Extraordinary valence : ≠4 (irregular point ○ )
Advantage of Subdivision (2)

- Piecewise smooth subdivision \[\text{[Hoppe '94]}\]
- Support more detail surfaces

- Smooth (s=0), dart (s=1), crease (s=2), and corner (s>2)

- Complex geometry
- Internal refinement of a mesh reduces consumption of bandwidth (bus, memory, and etc.)
## Subdivision Scheme Classification

<table>
<thead>
<tr>
<th></th>
<th>Face Split</th>
<th>Vertex Split</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular meshes</td>
<td>Quad. meshes</td>
<td>Doo-Sabin ($C^1$)</td>
</tr>
<tr>
<td>Approximating</td>
<td>Loop ($C^2$)</td>
<td><strong>Catmull-Clark ($C^2$)</strong></td>
</tr>
<tr>
<td>Interpolating</td>
<td>Modified Butterfly ($C^1$)</td>
<td>Kobbelt ($C^1$)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Biquartic($C^4$)</td>
</tr>
</tbody>
</table>

**Face Split**

**Vertex Split**
Subdivision Schemes

- **Approximating**
  - The limit curve does not lie on the vertices of the initial polygon because the vertices are discarded (or updated).

- **Interpolating**
  - Keep all the points from the previous subdivision step
Catmull-Clark Subdivision (1)

- Good results for most kinds of control mesh

- Most of modeling tools use Catmull-Clark subdivision
  - Autodesk 3ds Max
  - Autodesk Maya
  - PIXAR RenderMan

[Bolz '02]
Catmull-Clark Subdivision (2)

- Subdivision rules (regular point)

\[ f = \frac{v_1 + v_2 + v_3 + v_4}{4} \]
\[ e = \frac{v_1 + v_2 + e_1 + e_2}{4} \]
\[ v = \frac{prev. v + 2 \times \text{avg. } e' + \text{avg. } f}{4} \]
Adaptive Subdivision

- No generation of unnecessary vertices
- Improve performance with almost same quality

Uniform subdivision (Level = 4)

Adaptive subdivision (Max. level = 4)
LOD Selection

- Curvature
  - Flatness test
  - LOD is calculated from Max(D1, D2)

- Projected length (edge) or area (face)
Crack

- Adaptive subdivision has possibility to create cracks
- Cracks are created if each patch has different LOD
Crack Elimination

- Remove vertex

- Generate a face point & edge points
Why GPU?

- GPU has programmability enough for general computation
  - A programmable shader replaces a traditional fixed function unit as core processor
- GPU is faster than CPU for parallel processing of independent workloads
  - Integrated more arithmetic units (an arithmetic unit is simpler than that of CPU)
  - Enhanced matrix calculation (support dot product and multiply-and-add instructions)
  - Control path is optimized for non-data hazard workloads (efficient and simple)
Programmable Shader of GPU

- Traditional fixed-function unit → Programmable shader

Diagram showing the flow from CPU to GPU front end, primitive assembly, rasterization, raster operations, and frame buffer. Additional nodes include transform & lighting, texture unit, vertex shader, and pixel shader.
Parallel Processing

- Handle independent workloads
GPU Limitations

- **Program length limitation**
  - Maximum code length is limited.
  - Shader program switching overhead is very heavy.
  - But this problem can be solved at the next version of shader model.

- **Weak data feedback**
  - Optimized for unidirectional data flow (input-to-framebuffer)
  - Some extensions support data feedback features but limited.
Previous Works (1)

  - CPU implementation using SIMD instruction
  - Pre-computation of tables for all depth and valences
    - Poor flexibility and large tables
  - Adaptive subdivision
  - Final subdivided vertices send to GPU
    - No gain of CPU-to-GPU data transfer bandwidth
Previous Works (2)

  - GPU implementation version of their previous work
  - Final subdivided vertices send to CPU and re-send to GPU for rendering
    - The data should be sent to vertex shader input for rendering, but there was no path from frame buffer or texture memory to vertex shader in that time
- Pixel shader program on GPU for subdivision
- Adaptive subdivision using flatness test at each level
- CPU read the flatness test results from the video memory and decides which patches need further tessellation for adaptive subdivision
Previous Works (3) – cont’d

- All patches are subdivided by only one level at every subdivision iteration
  - Good locality between a patch and its neighbors
  - Poor locality between a current patch and the same patch of the next iteration
- Use copy-to-texture for feedback of the intermediate data
Previous Works (4)

  - Regular processing using fragment mesh
    - Irregular point is placed at center
    - 1-ring regular point meshes are overlapped
Previous Works (4) – cont’d

- Processing of irregular points causes inefficient memory access and shader context switching (regular point shader program and irregular point shader program)
  - All fragment meshes have regular pattern
    - 1-irregular point & regular points
    - Can be used of united shader program
- Few information about adaptive subdivision
Previous Works (5)

  - Exploration for many new memory access features in OpenGL API extension
  - Using frame buffer object (FBO)
Previous Works (5) – cont’d

● Using vertex buffer object (VBO)

● Or vertex texture
Problems

- Context switching is large overhead
  - FBO destination switching (frame buffer or texture memory)
  - Multiple shader program switching
  - CPU (host) should handle both context switching

- Neighbor mesh information is overlapped
  - Redundant information
Problems – cont’d

- Missing temporal locality at each subdivision step
  - Flatness test at every subdivision steps
    - Crack should be eliminated at final subdivision step
  - Breath first operation (Subdivision step 1 of patch 1 → subdivision step 1 of patch 2 → ... → subdivision step n of patch 1)

  ![Diagram](image)

  - step 1
  - step 2
  - can be reused at next step
  - but flushed in the cache
Question?

Thank You!