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

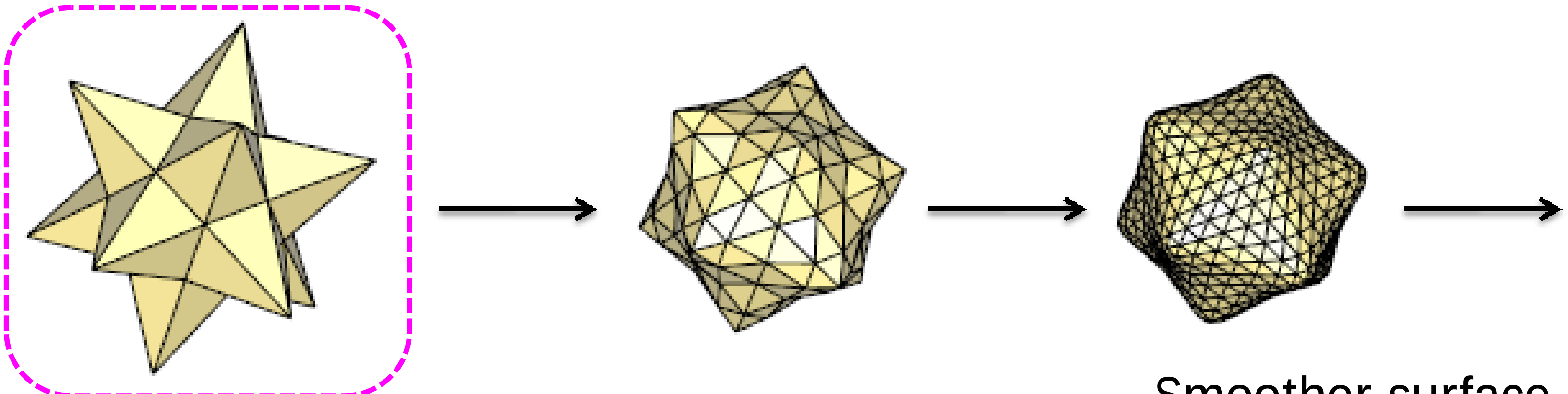
Introduction

- 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)

KAIST

Subdivision Meshes (1)

- **Recursively refine a polygonal mesh**



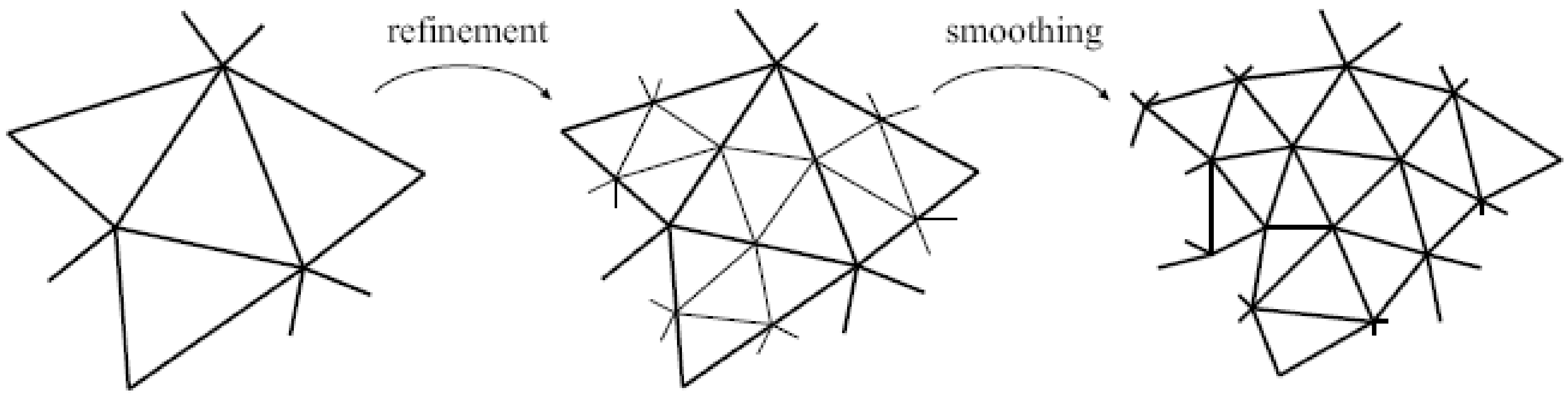
Original control mesh

Smoother surface
by recursive processing

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

Subdivision Meshes (2)

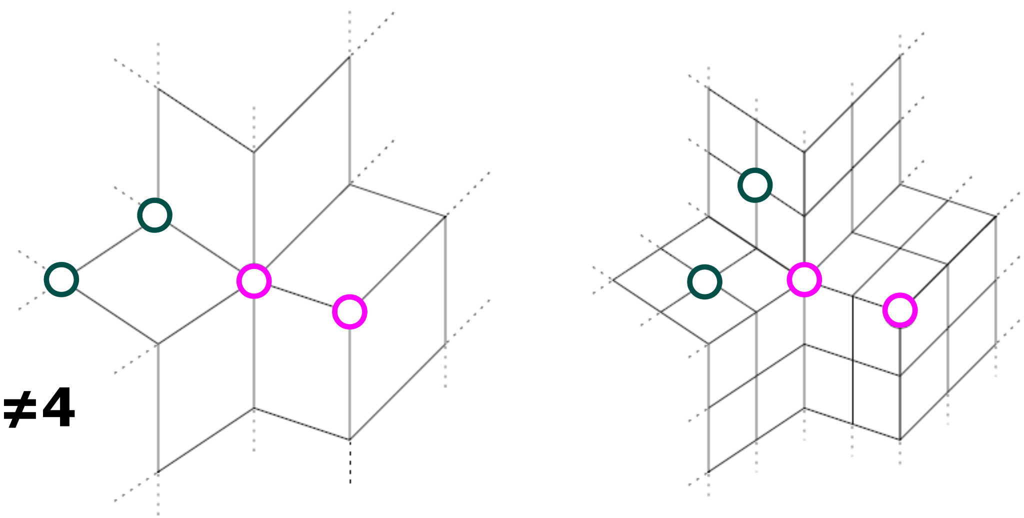
- **Two phase process**
  - **Refinement phase: creates new vertices and reconnects to create new triangles**
  - **Smoothing phase: computes new positions for the vertices**



Advantage of Subdivision (1)

- Efficiency
  - Modeling is easy
- Arbitrary topology
  - Classic spline approaches have great difficulty with control meshes of arbitrary topology.

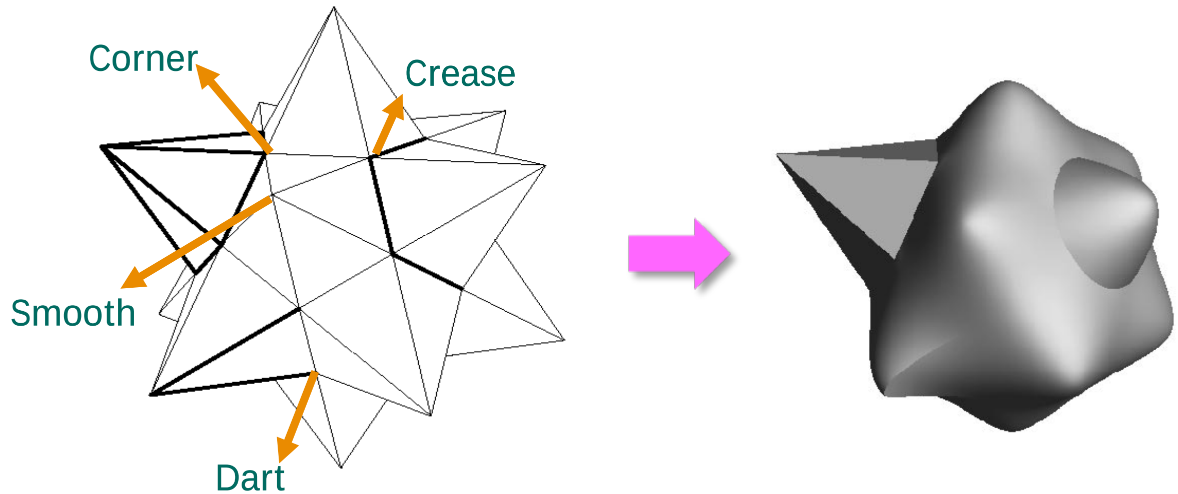
Standard valence : 4
(regular point ○)
Extraordinary valence : ≠4
(irregular point ○)



KAIST

Advantage of Subdivision (2)

- **Piecewise smooth subdivision *[Hoppe '94]***
  - **Support more detail surfaces**

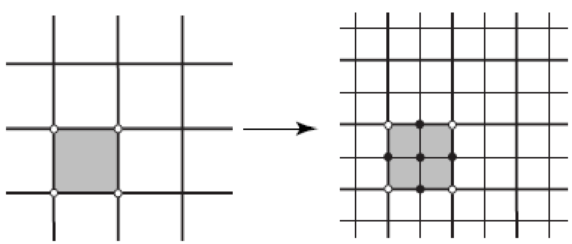


smooth (s=0), dart (s=1), crease (s=2), and corner (s>2)

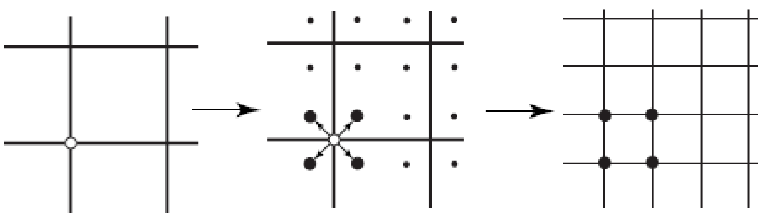
- **Complex geometry**
  - **Internal refinement of a mesh reduces consumption of bandwidth (bus, memory, and etc.)**

Subdivision Scheme Classification

Face Split			Vertex Split
	Triangular meshes	Quad. meshes	Doo-Sabin (C^1)
Approximating	Loop (C^2)	**Catmull-Clark (C^2)**	Midedge (C^1)
Interpolating	Modified Butterfly (C^1)	Kobbelt (C^1)	Biquartic(C^4)



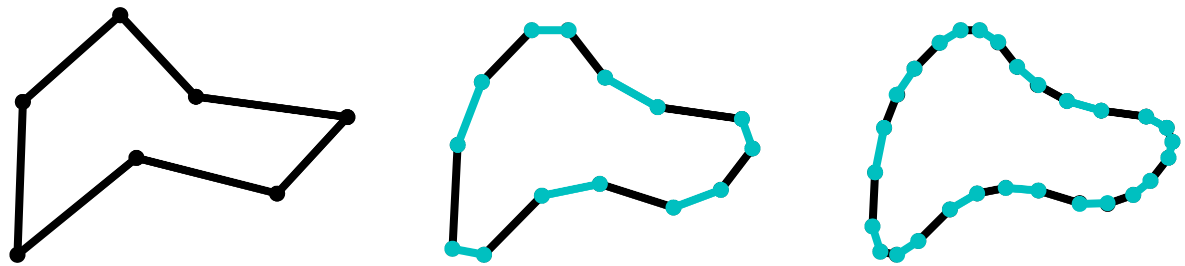
Face Split



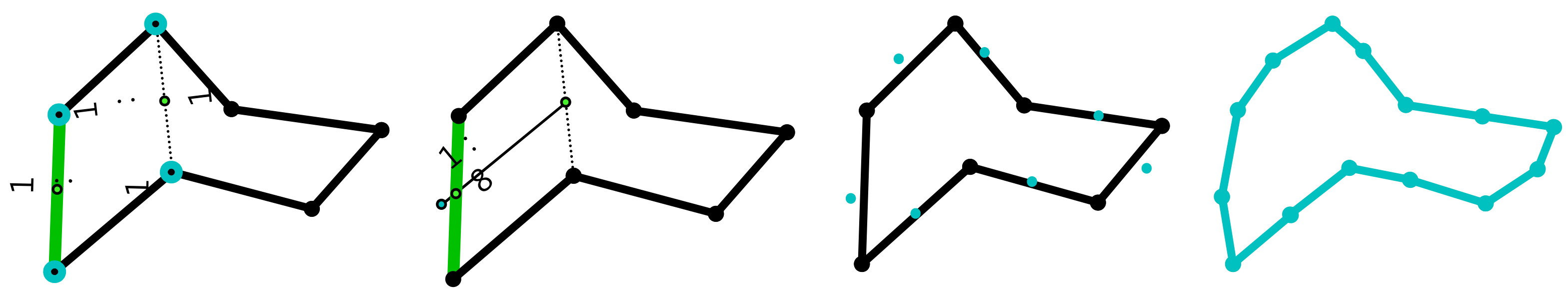
Vertex Split

Subdivision Schemes

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

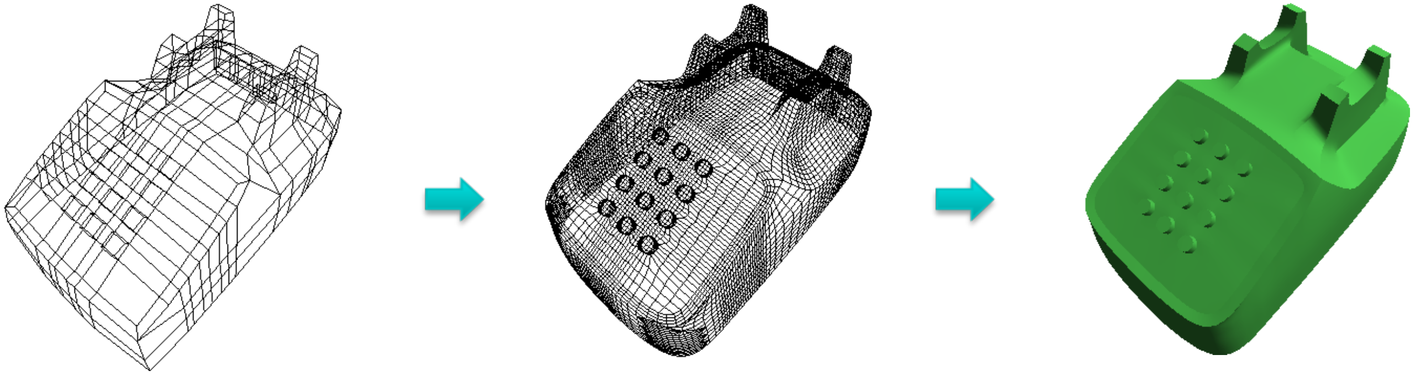


- *Interpolating*
  - **Keep all the points from the previous subdivision step**



Catmull-Clark Subdivision (1)

- Good results for most kinds of control mesh

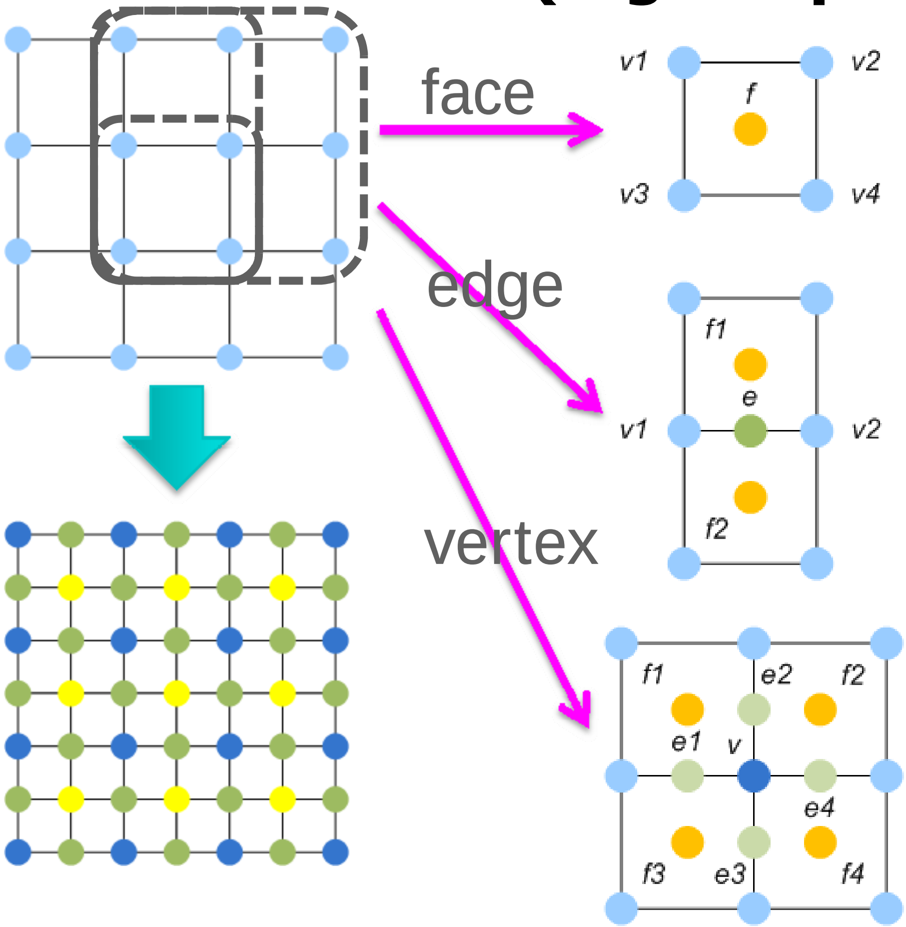


[Bolz '02]

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

Catmull-Clark Subdivision (2)

- **Subdivision rules (regular point)**



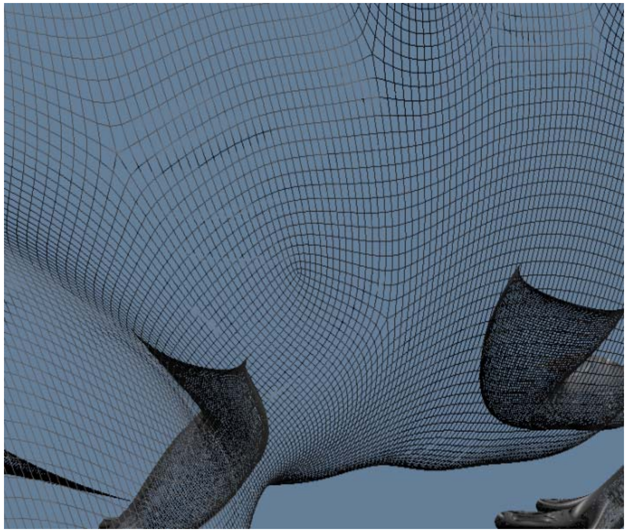
$$f = \frac{v1 + v2 + v3 + v4}{4}$$

$$e = \frac{v1 + v2 + f1 + f2}{4}$$

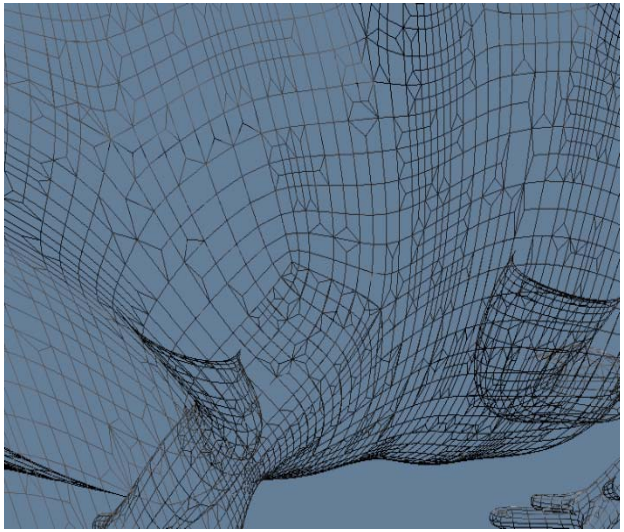
$$v = \frac{prev.\, v + 2 \times avg.\, e' + avg.\, f}{4}$$

Adaptive Subdivision

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



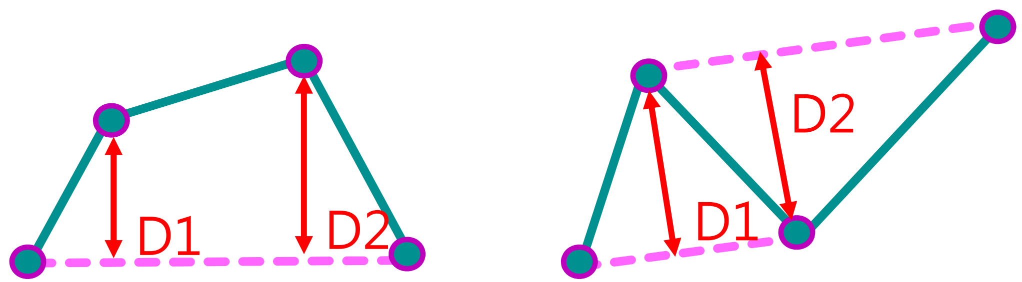
Uniform subdivision (Level = 4)



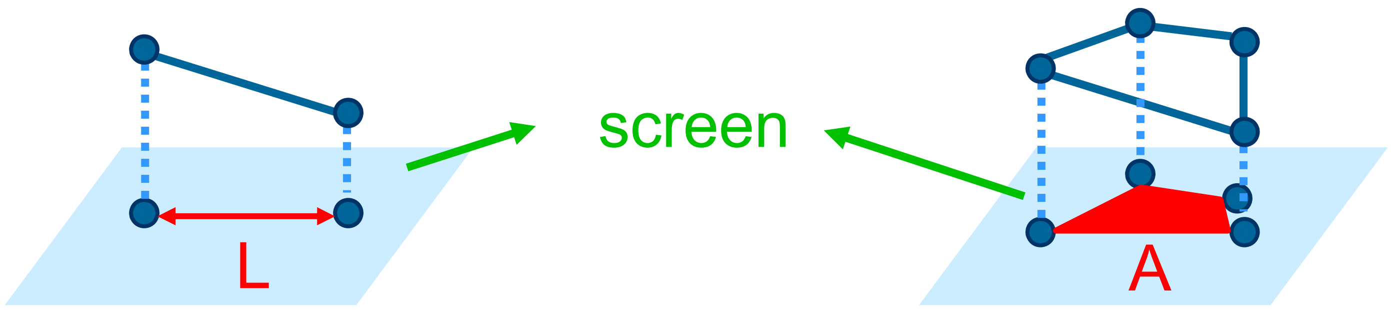
Adaptive subdivision (Max. level = 4)

LOD Selection

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

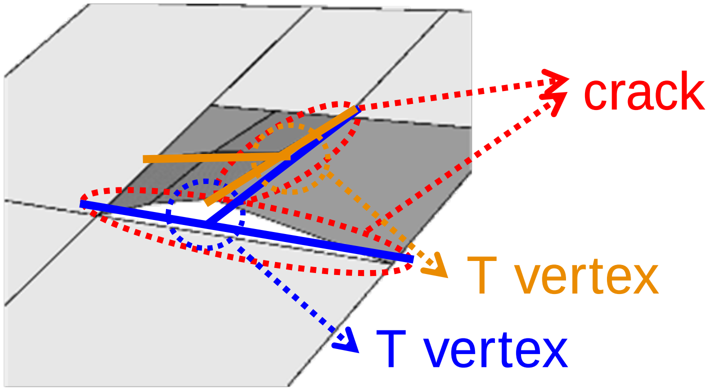


- **Projected length (edge) or area (face)**



Crack

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



Crack Elimination

- **Remove vertex**

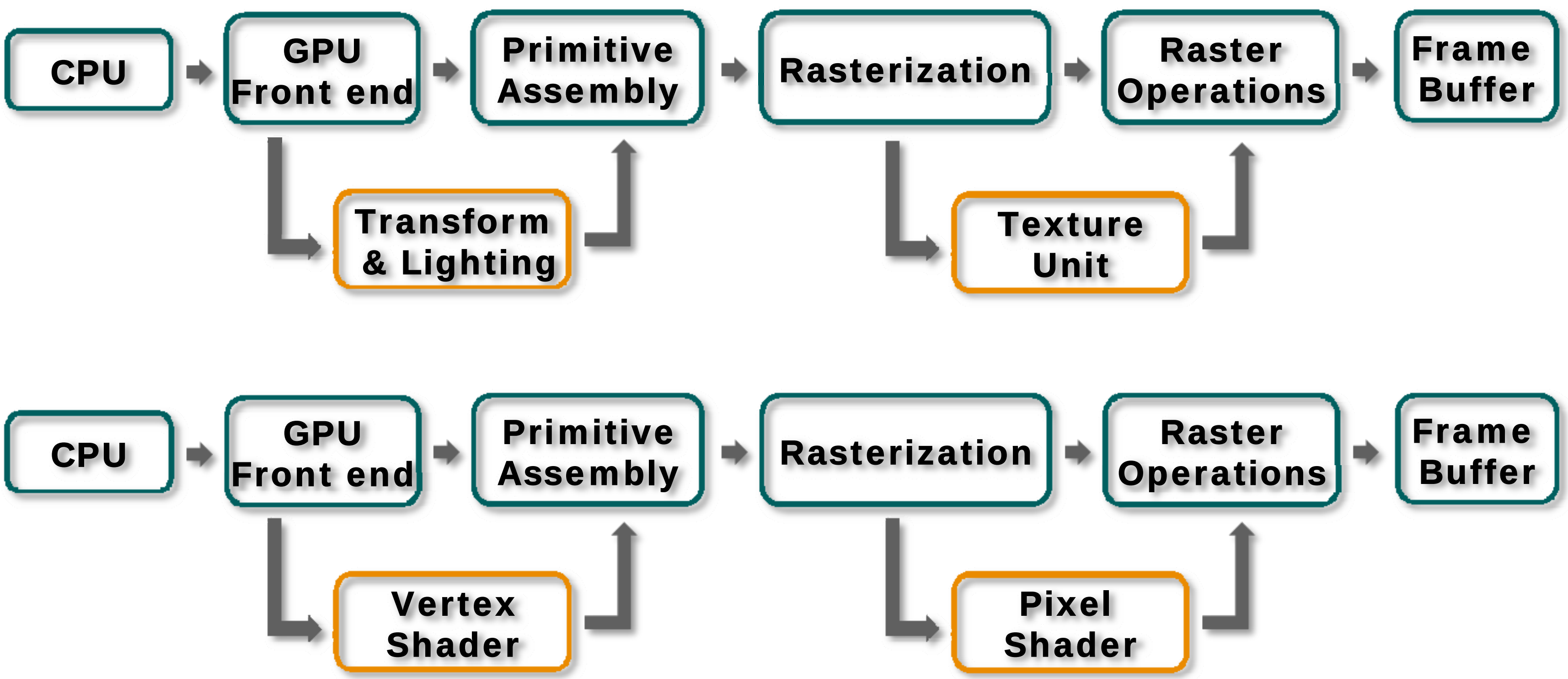
- **Generate a face point & edge points**

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)**

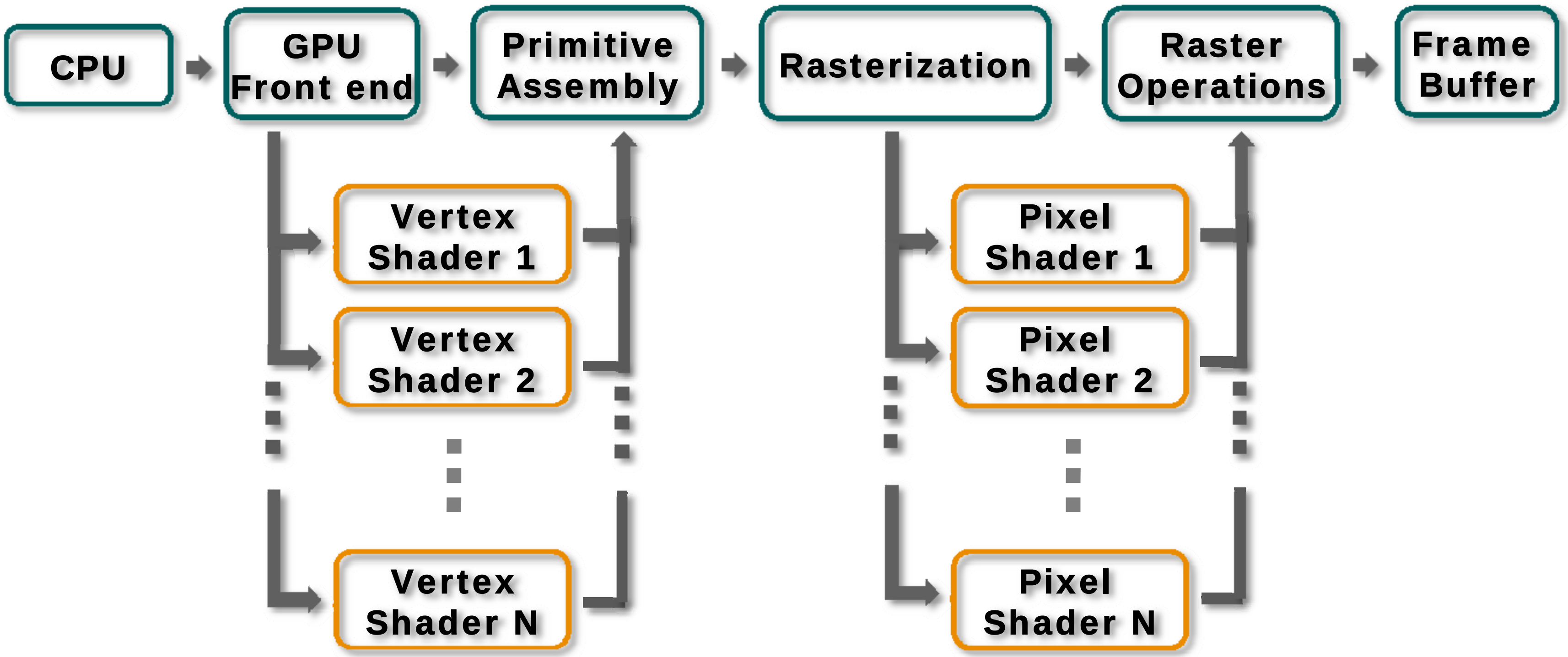
Programmable Shader of GPU

- **Traditional fixed-function unit → Programmable shader**



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 feed back features but limited.**

Previous Works (1)

- **Bolz, J. and Schröder, P. 2002. Rapid Evaluation of Catmull-Clark Subdivision Surfaces**
  - **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**

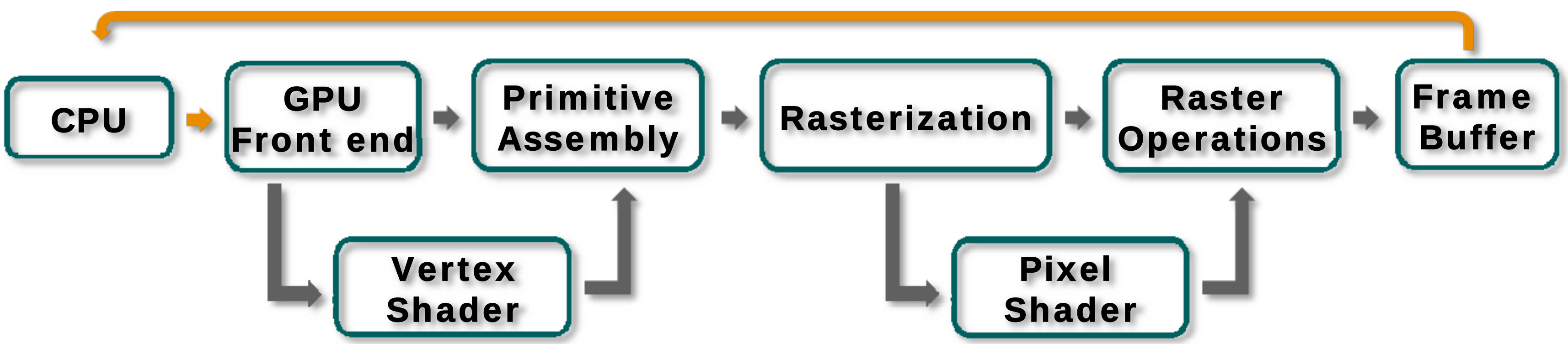
Previous Works (2)

- **Bolz, J. and Schröder, P. 2002. Evaluation of subdivision surfaces on programmable graphics hardware**
  - **GPU implementation version of their previous work**
  - **Final subdivided vertices send to CPU and re-send to GPU for rendering**
    - **The data shoud 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**

KAIST

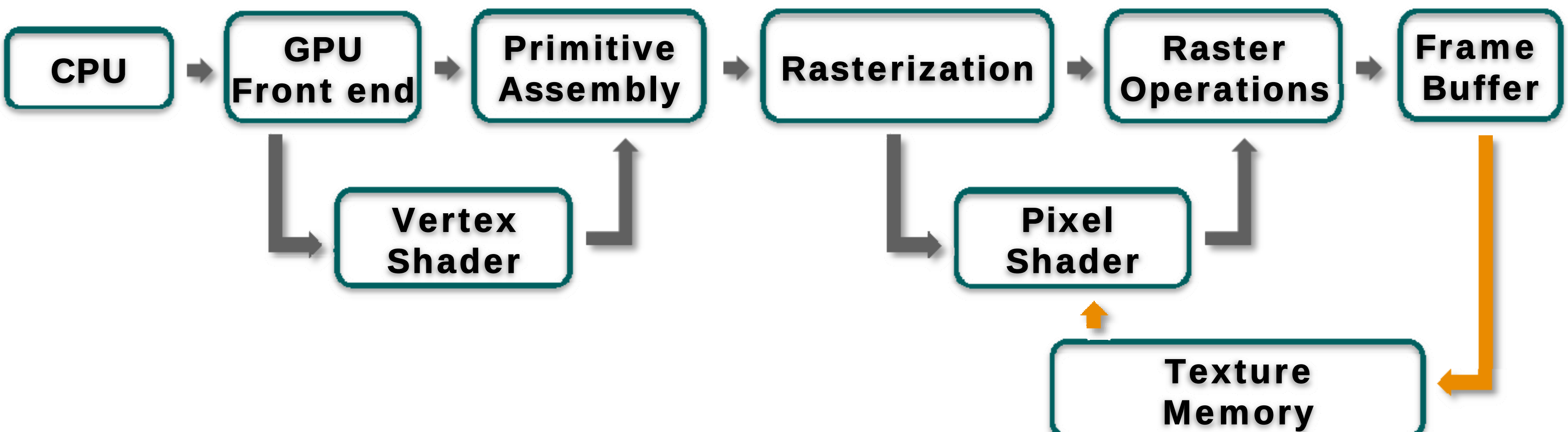
Previous Works (3)

- Bunnel. 2005. Adaptive Tessellation of Subdivision Surfaces with Displacement Mapping (GPU Gem2)
  - 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



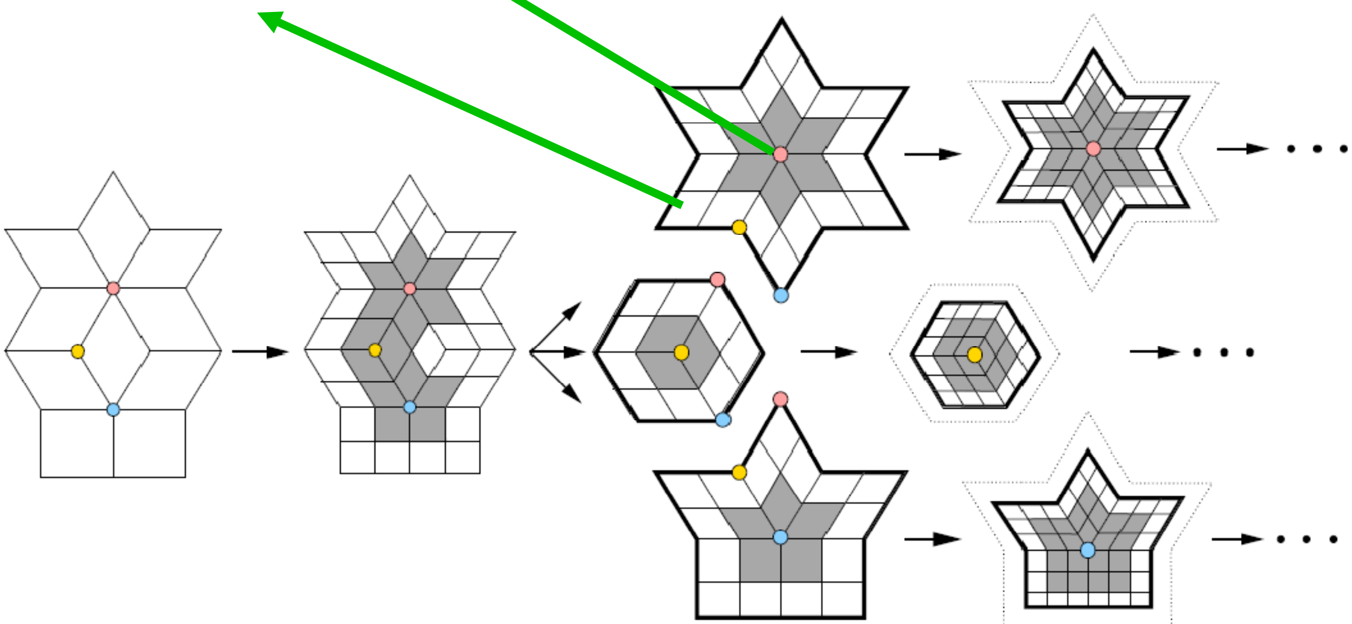
Previous Works (3) – cont'd

- 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)

- **Le-Jeng Shiue 2005. A Real-time GPU subdivision Kernel**
  - **Regular processing using fragment mesh**
    - **Irregular point is placed at center**
    - **1-ring regular point meshes are overlapped**

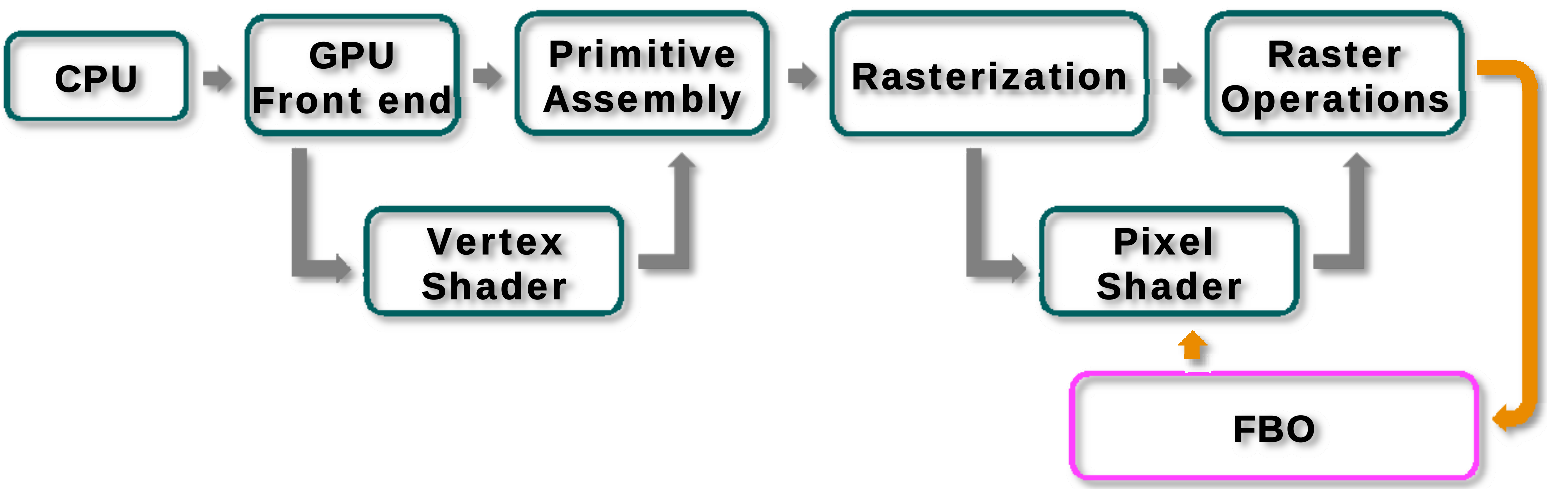


Previous Works (4) – cont'd

- 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

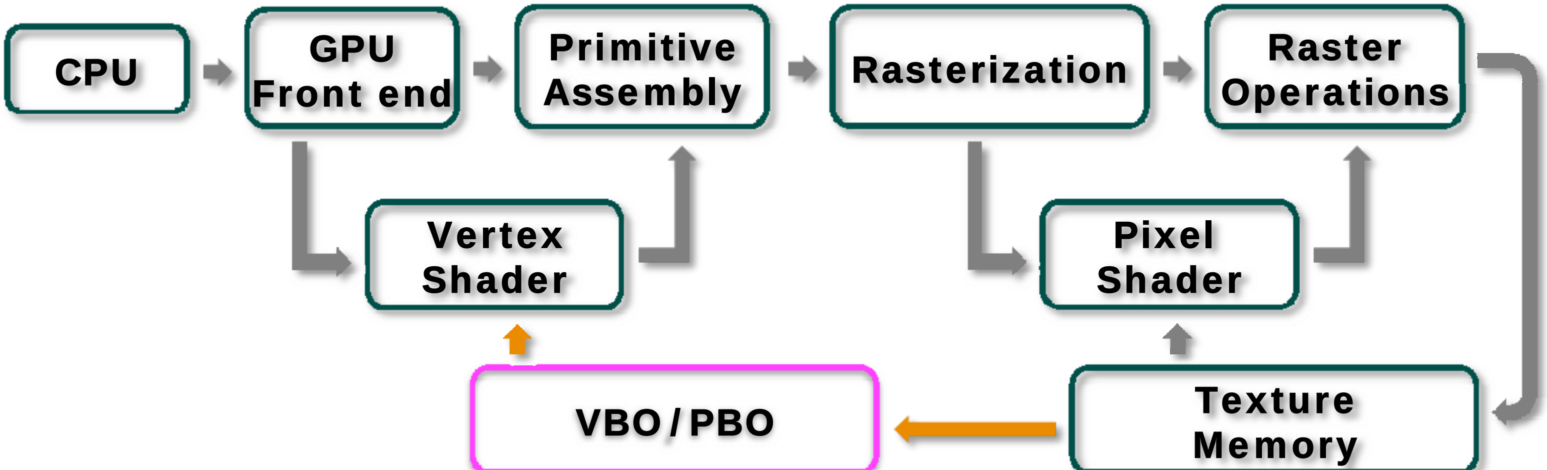
Previous Works (5)

- **Minho Kim. 2005. Real-time Loop Subdivision on the GPU**
  - **Exploration for many new memory access features in OpenGL API extension**
    - **Using frame buffer object (FBO)**

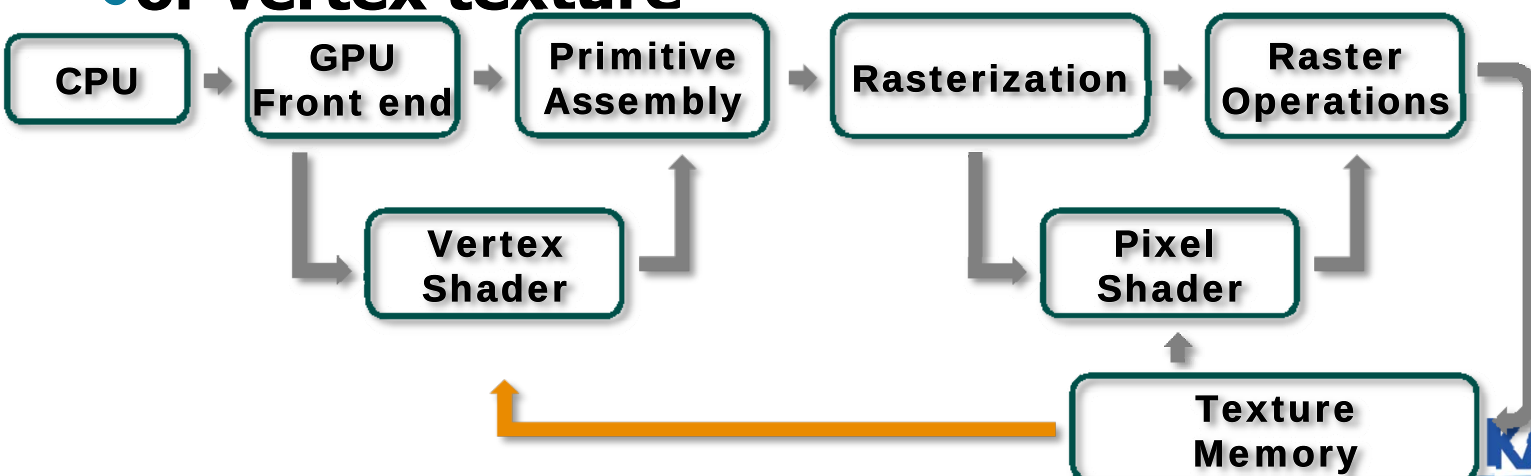


Previous Works (5) – cont'd

- Using vertex buffer object (VBO)



- or vertex texture



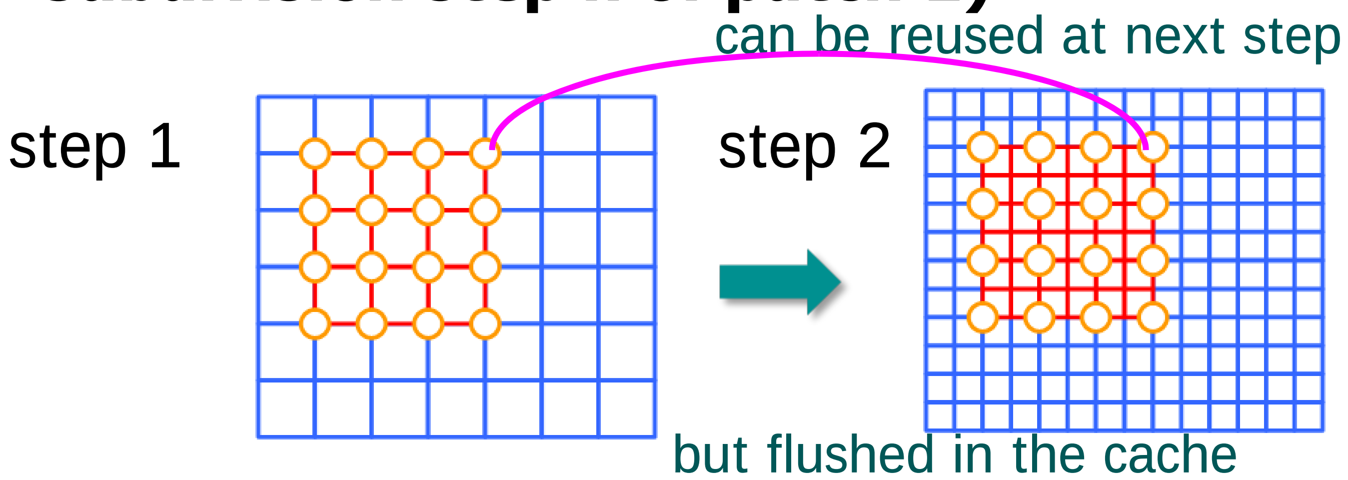
Problems

- **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**

KAIST

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)



Question?

Thank You!