GPU TECHNOLOGY CONFERENCE

NANYANG TECHNOLOGICAL UNIVERSITY

MAGIC
Multi-plAtform Game Innovation Centre

Adaptive NURBS Tessellation with CUDA

Presenter: Dr. Qiu Jie

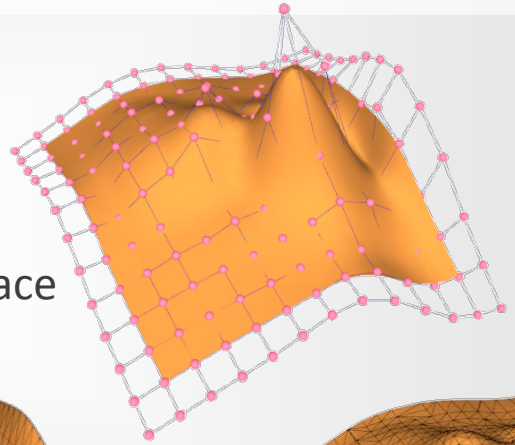
Multi-plAtform Game Innovation Centre (MAGIC), Nanyang Technological University (NTU), Singapore

About MAGIC

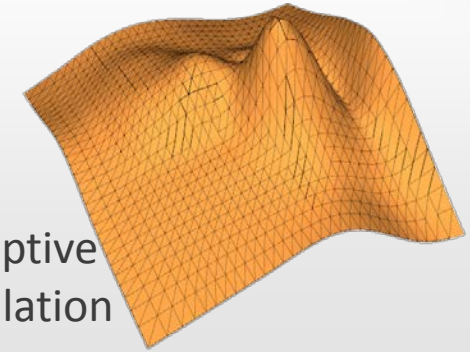
- Established in Nanyang Technological University (NTU), Singapore on 1 Nov 2012
- Cross-disciplinary research with 26 professors, 33 researchers, 11 PhD
- To translate scientific ideas into technological products and services
- To increase the capability of Future Studios in media & entertainment
- Launched Future Studios Research Lab (FSR Lab) on 21 Jan 2014

Adaptive NURBS Tessellation

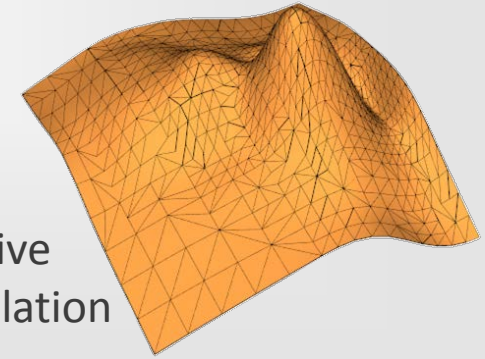
- Tessellation
  - Convert NURBS to a set of polygons
- Adaptive tessellation
  - According to
    - Curvature
    - View (camera distance to object)



NURBS surface

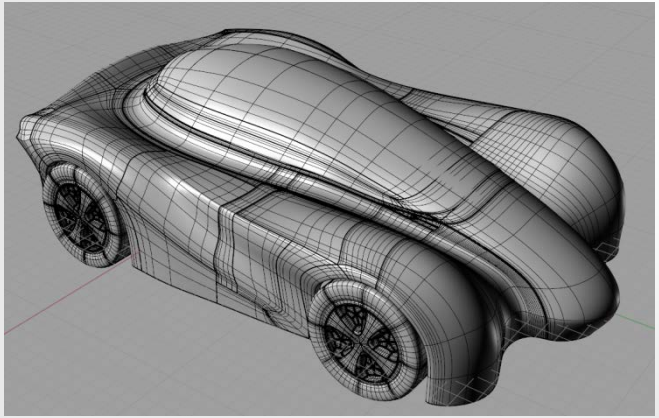


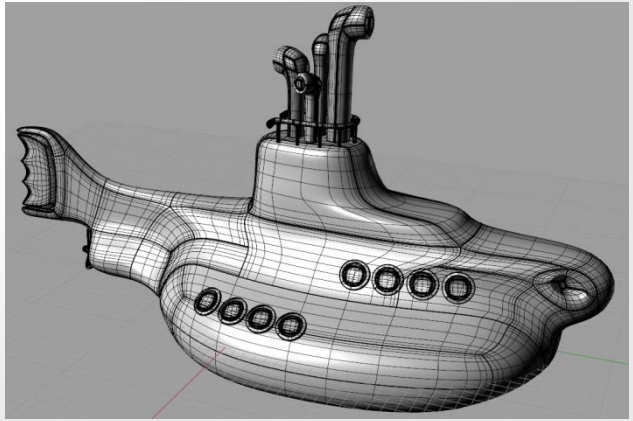
Non-adaptive tessellation



Adaptive tessellation

Tessellating NURBS surface

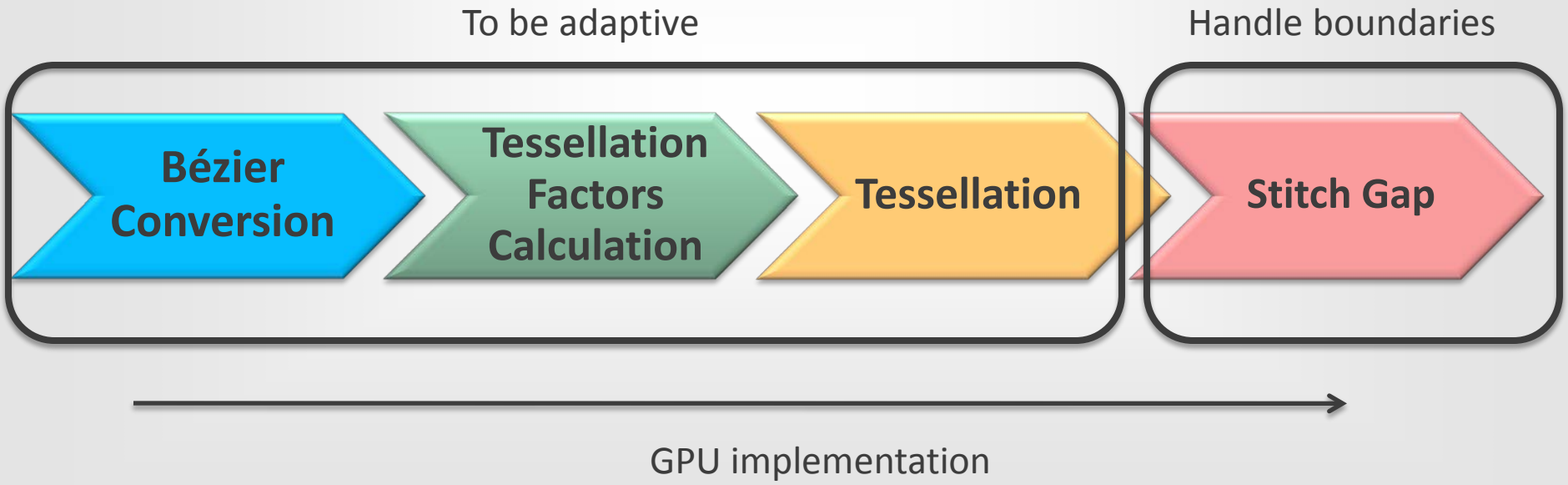




Adaptive NURBS Tessellation

- How to be adaptive?
  - Divide the surface into patches
  - Choose accurate, effective criterion to tessellate each patch
- How to handle the connection of neighbor parts?
  - Extract neighbour info of each patch to check boundary consistency
  - Stitch gaps if not consistent
- How to speed up?
  - Implementation on GPU

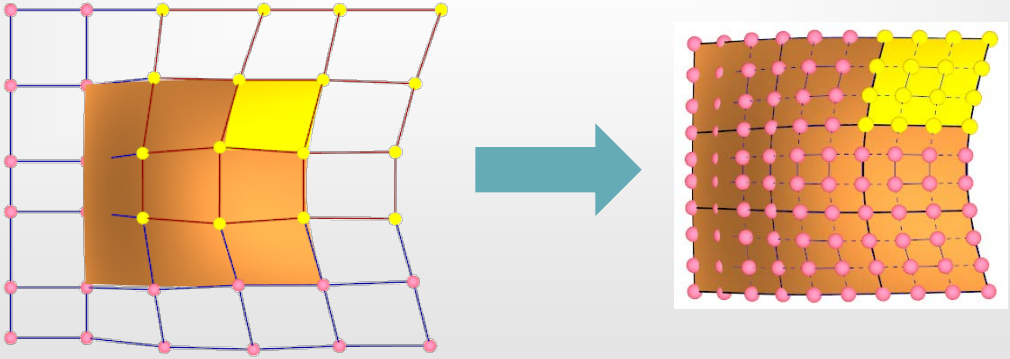
Proposed Framework



Bézier Conversion

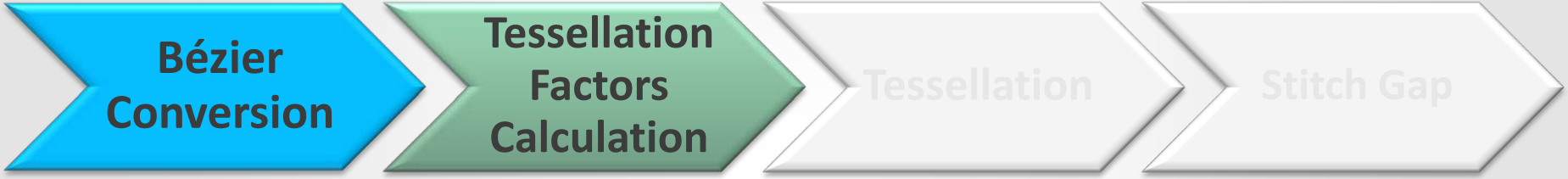
Bézier Conversion → Tessellation Factors Calculation → Tessellation → Stitch Gap

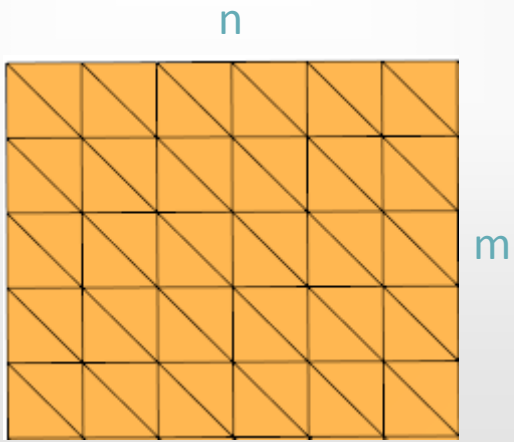
convert *NURBS surface* to *a set of Bézier patches*



More accurate for tessellation factor calculation

Tessellation Factors Calculation





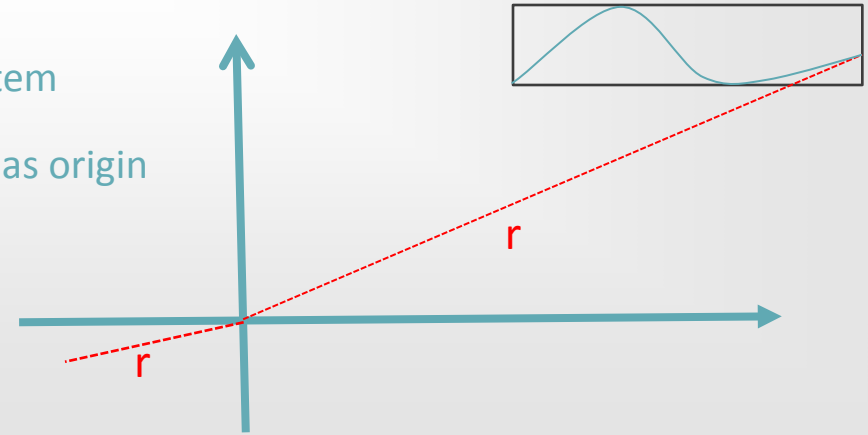
Tessellation Factors Calculation

- One traditional method

$$tess \propto \max_i \left(\left\| \Delta^2 (w_i P_i) \right\| + (r - \varepsilon) \left| \Delta^2 w_i \right| \right)$$

P_i: control points
w_i: weights of control points

  - Related to the origin of the coordinate system
  - Heuristic method: Center of bounding box as origin



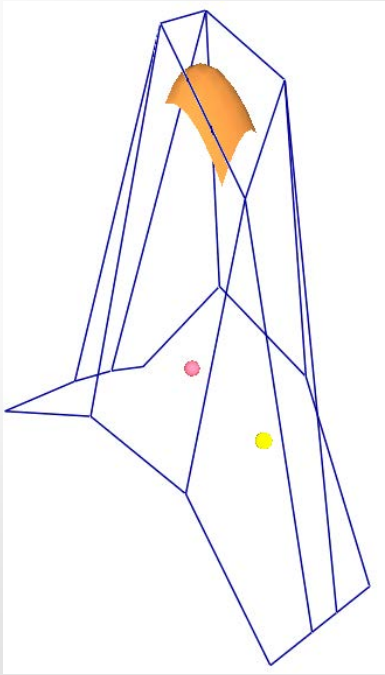
Tessellation Factors Calculation

- Our method
  - Using a new objective function to find origin O

$$\min_{O}\left(\sum_{j=0}^{n}(P_j - O)^2 + \sum_{i=0}^{n-2}\left|\Delta^2 w_i\right|\left(\frac{\Delta^2 w_i P_i}{\Delta^2 w_i} - \mathrm{O}\right)^2\right)$$

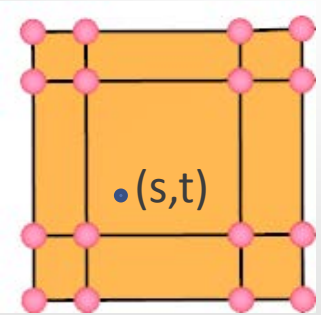
Model	Old method (ε =0.1)	Our method (ε =0.1)
1	42	36
2	142	120
3	417	360

# of vertices



Tessellation

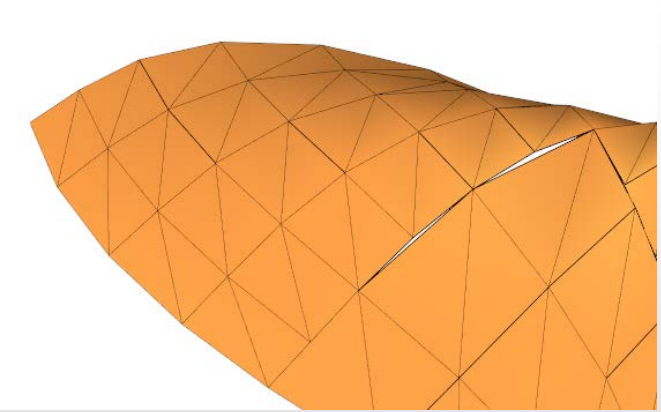
Bézier Conversion → Tessellation Factors Calculation → Tessellation → Stitch Gap

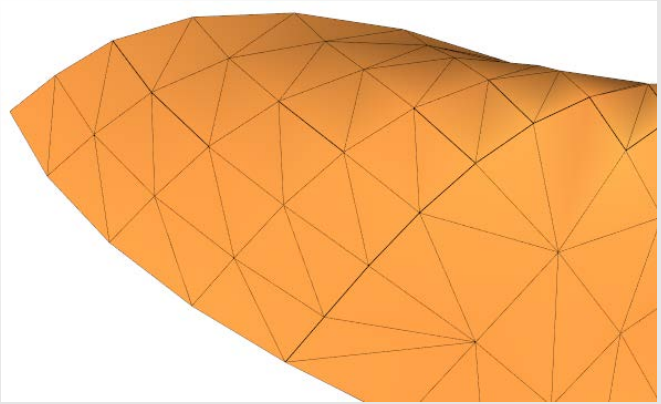


$$S(s,t) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} P_{i,j} B_i(s) B_j(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} B_i(s) B_j(t)}$$

Stitch Gap

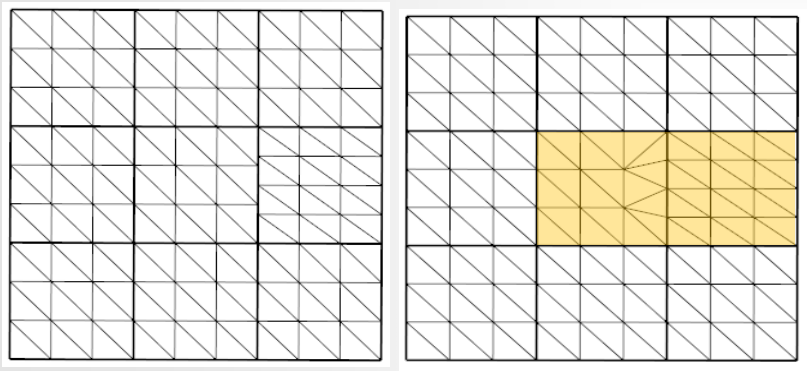
Bézier Conversion → Tessellation Factors Calculation → Tessellation → Stitch Gap



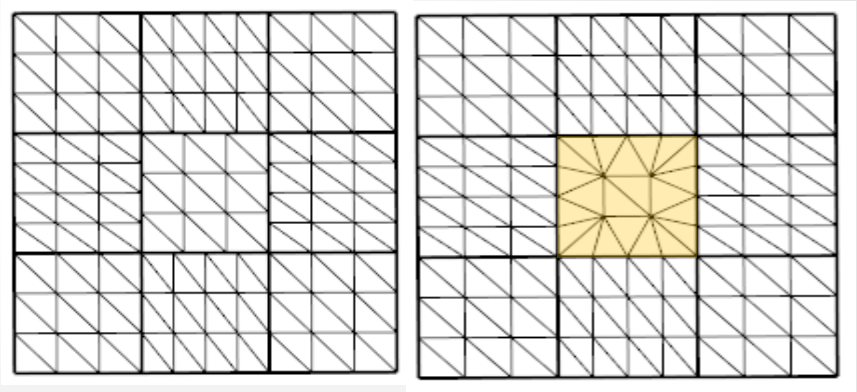


Problem: inconsistent tessellation factors between patches gap

Stitch Gap



1 side

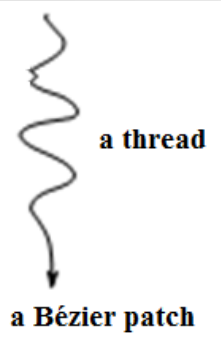


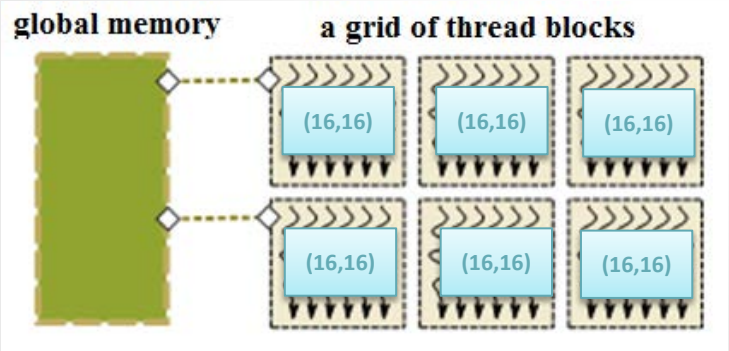
4 sides

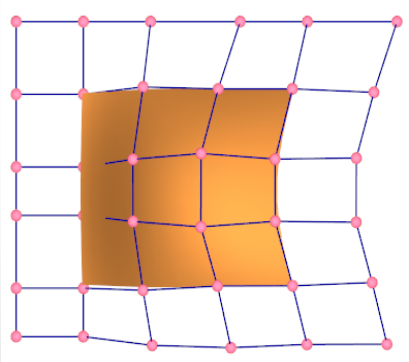
Re-design boundary connections

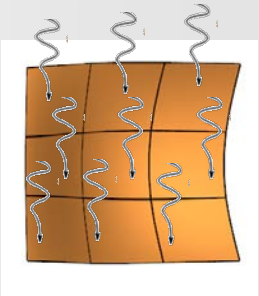
Implementation on GPU – level of parallelism

- Thread assignment

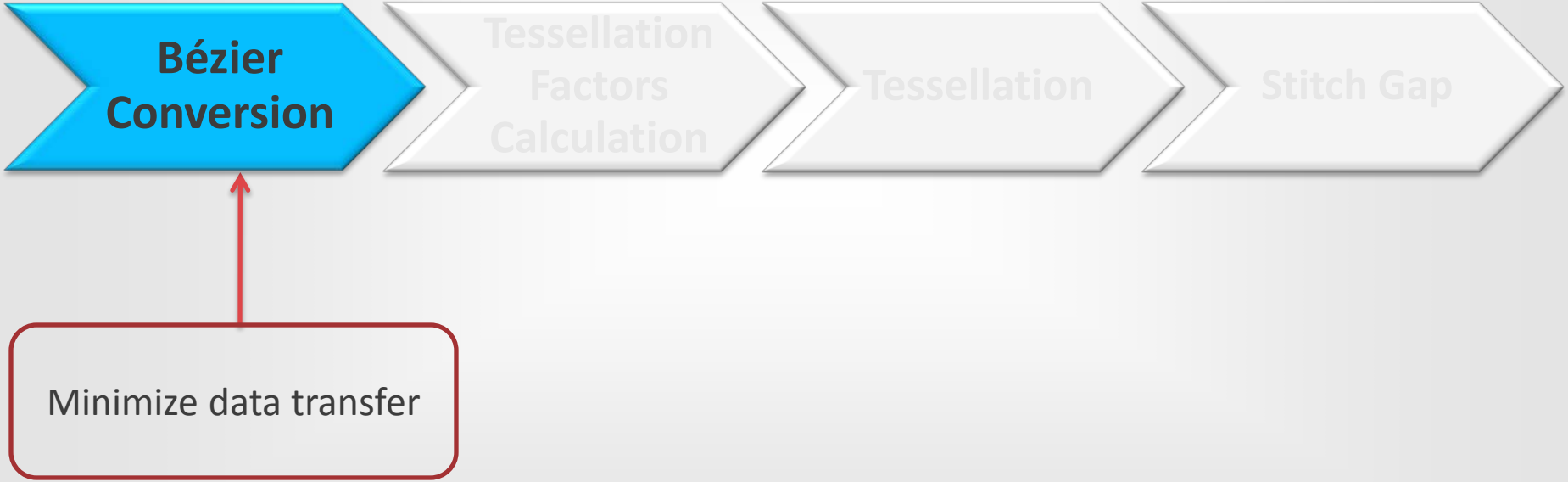






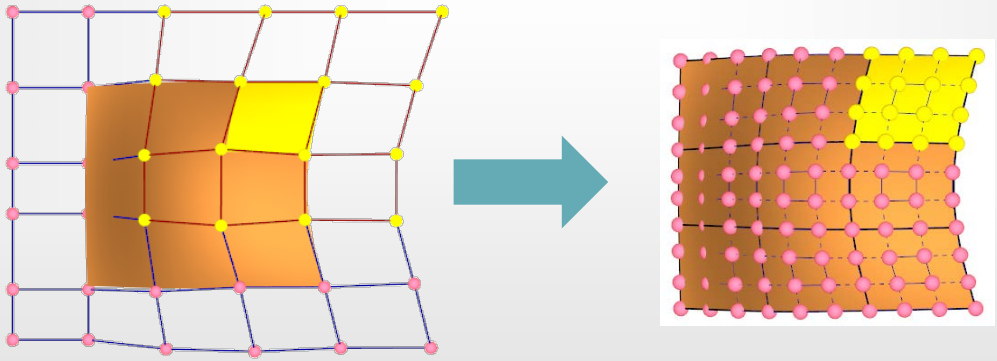


Implementation on GPU – data management



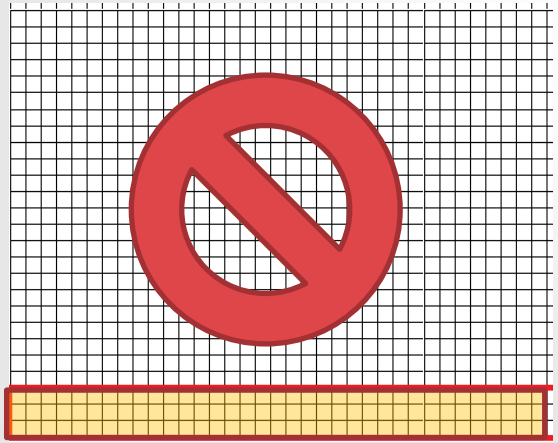
Implementation on GPU

- NURBS surface to Bézier patches
  - Compute Bézier control points

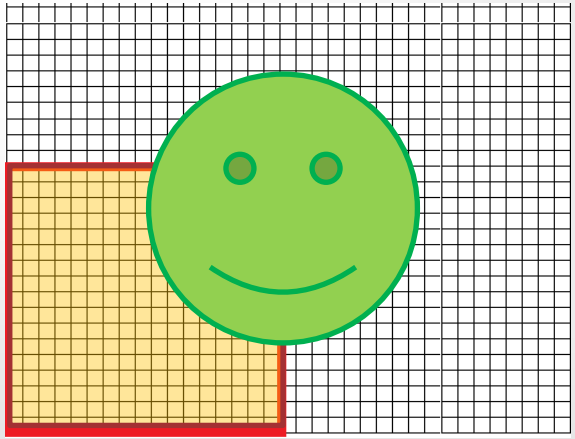


Implementation on GPU

- Copy control points needed by each block to shared memory
  - Avoid duplicate memory read for global memory

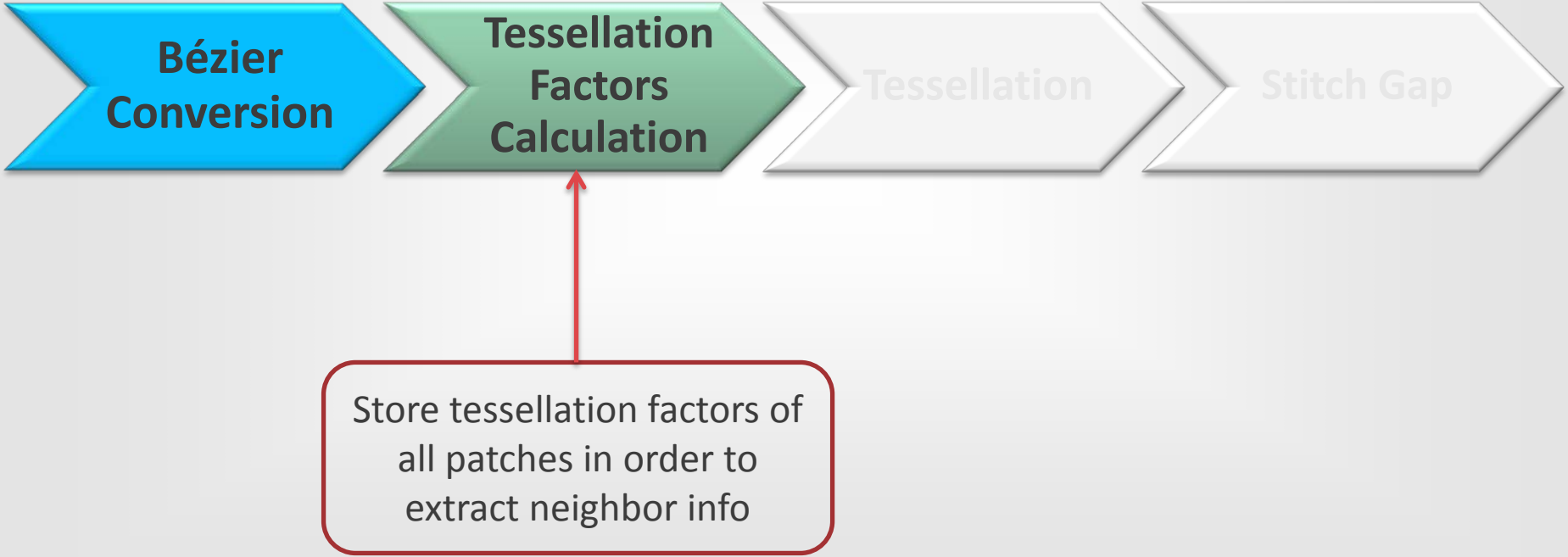


4 X 259 = 1036 points for a block



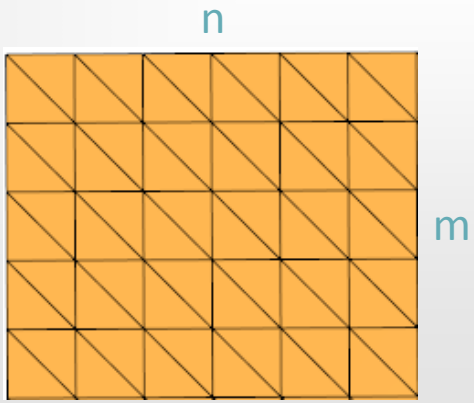
19 X 19 = 361 points for a block

Implementation on GPU



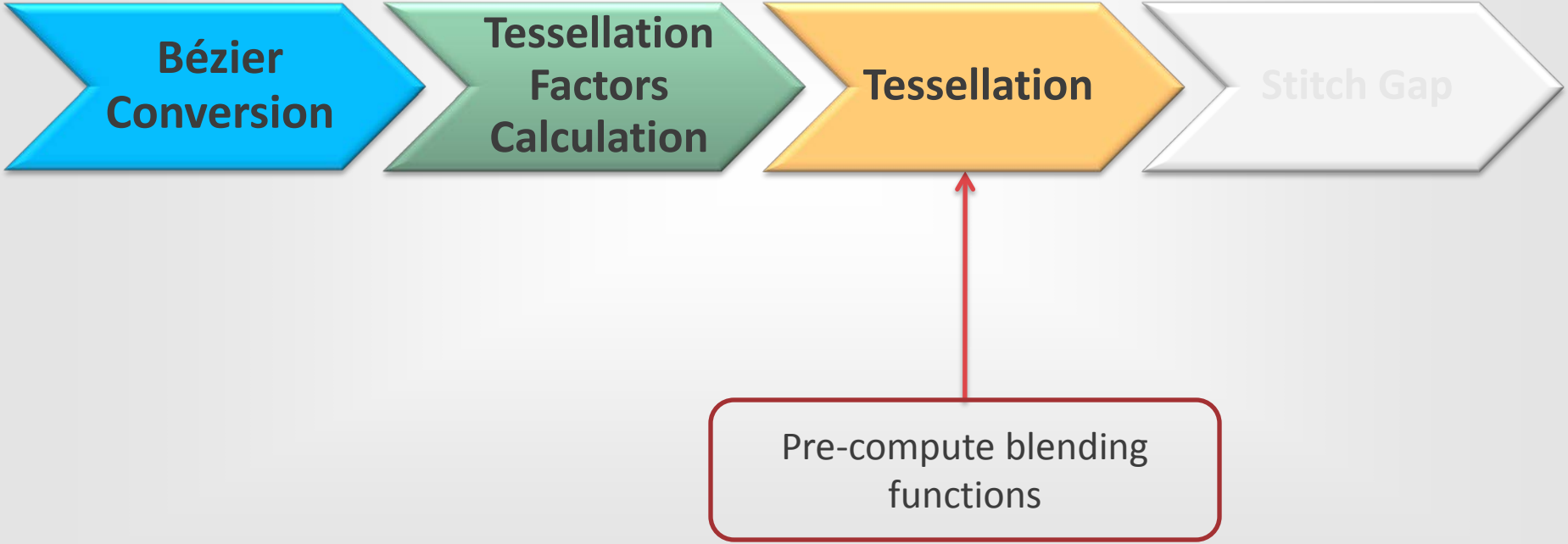
Implementation on GPU

- Calculate tessellation factors
  - Calculate tessellation factors (2 directions)
  - Store them in global memory



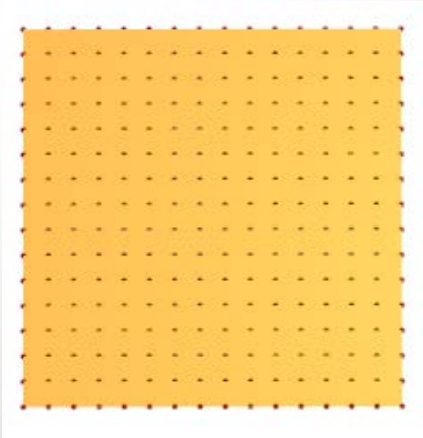
Patch ID	Direction 1	Direction 2
0	h_0	v_0
1	h_1	v_1
2	h_2	v_2
…	…	…
…	…	…
k	h_k	v_k

Implementation on GPU



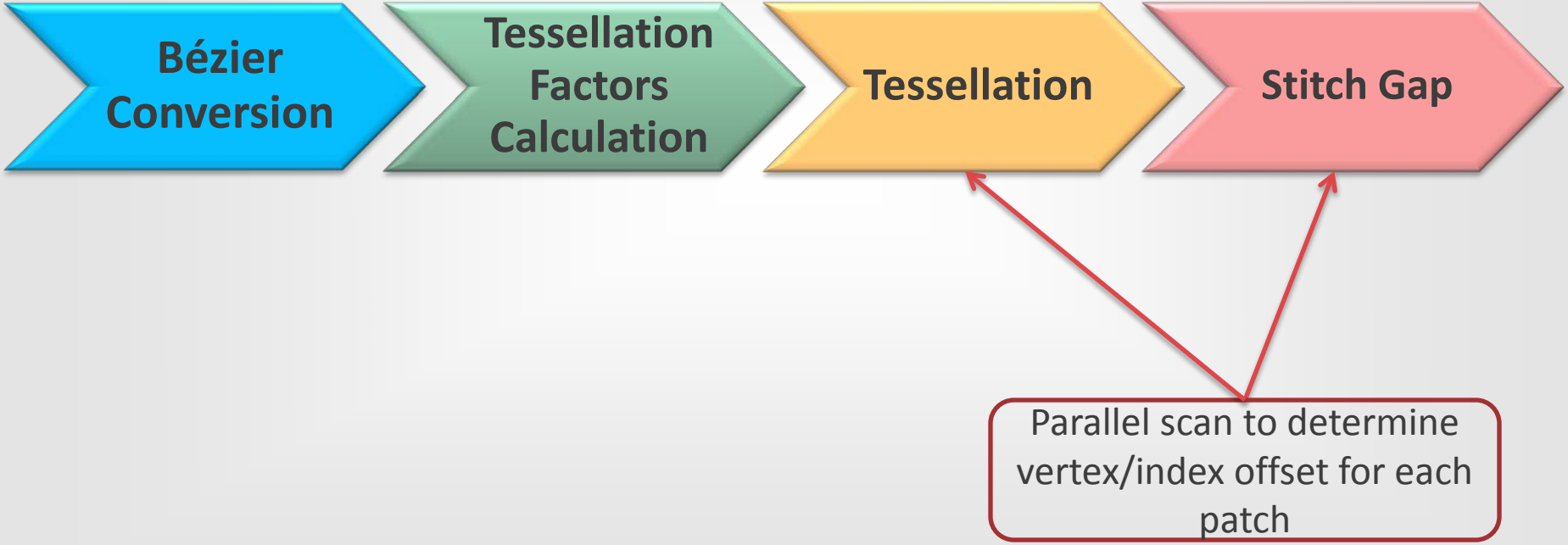
Implementation on GPU

- Tessellation using regular pattern
  - Fast evaluation
    - Each row has the same $B_i(s)$
    - Each column has the same $B_j(t)$



$$S(s,t) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} P_{i,j} B_i(s) B_j(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} B_i(s) B_j(t)}$$

Implementation on GPU



Implementation on GPU

- Vertex offset
  - Determine # of vertices for each patch based on tessellation factors
  - Determine vertex offset for each patch

Patch ID	# vertices
0	$(\#v)_0$
1	$(\#v)_1$
2	$(\#v)_2$
…	…
k	$(\#v)_k$

Prefix sum →

vertex offset
0
$(\#v)_0$
$(\#v)_0+(\#v)_1$
…
$SUM((\#V)_0, (\#V)_{k-1})$

Implementation on GPU

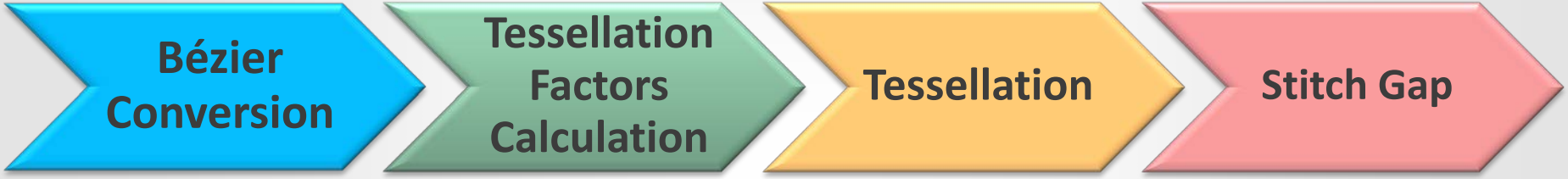
- Transition region and index offset
  - Determine # of indices on each patch (include transition regions)
  - Determine index offset for each patch

Patch ID	# indices
0	$(\#id)_0$
1	$(\#id)_1$
2	$(\#id)_2$
…	…
k	$(\#id)_k$

Prefix sum →

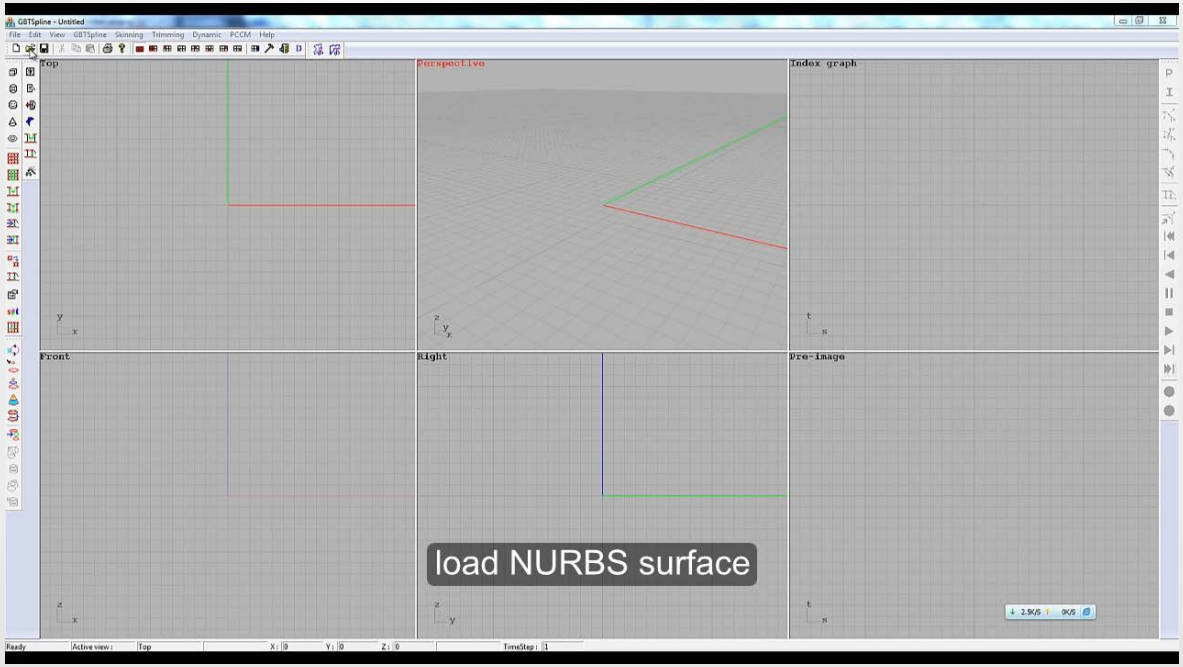
index offset
0
$(\#id)_0$
$(\#id)_0 + (\#id)_1$
…
$SUM((\#id)_0, (\#id)_{k-1})$

Implementation on GPU

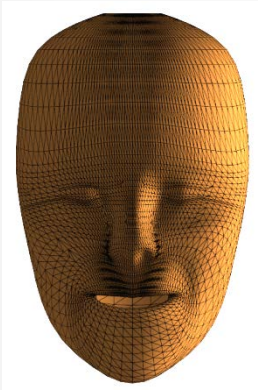


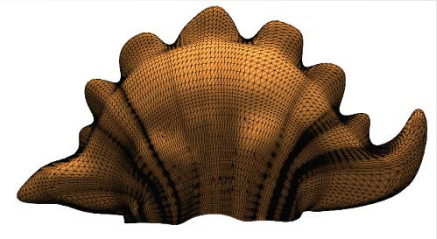
- Minimize data transfer
- Store patch info for easy neighbour info extraction
- Regular tessellation pattern
- Parallel scan

Video Demonstration



Results

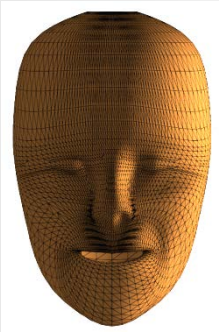


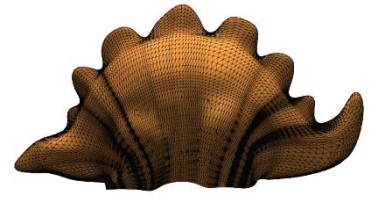


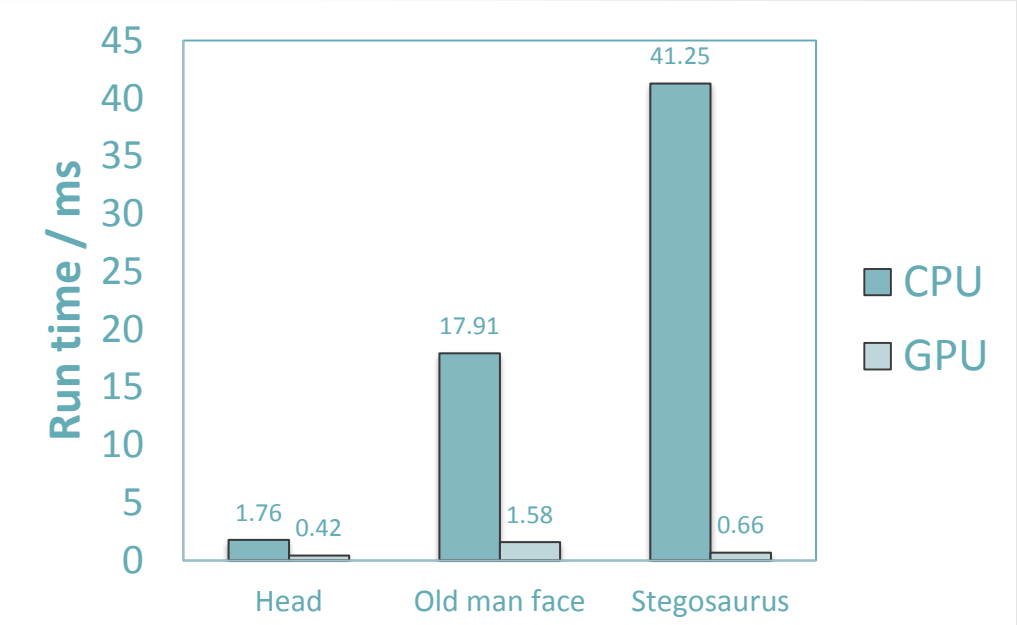
	Head	Old man face	Stegosaurus
Patch No.	360	3828	12312
Triangle No.	2298	10216	25530

Results

- CPU (intel i7-2600 3.4GHz) vs GPU (GTX590 512 CUDA cores 160GB/s bandwidth)

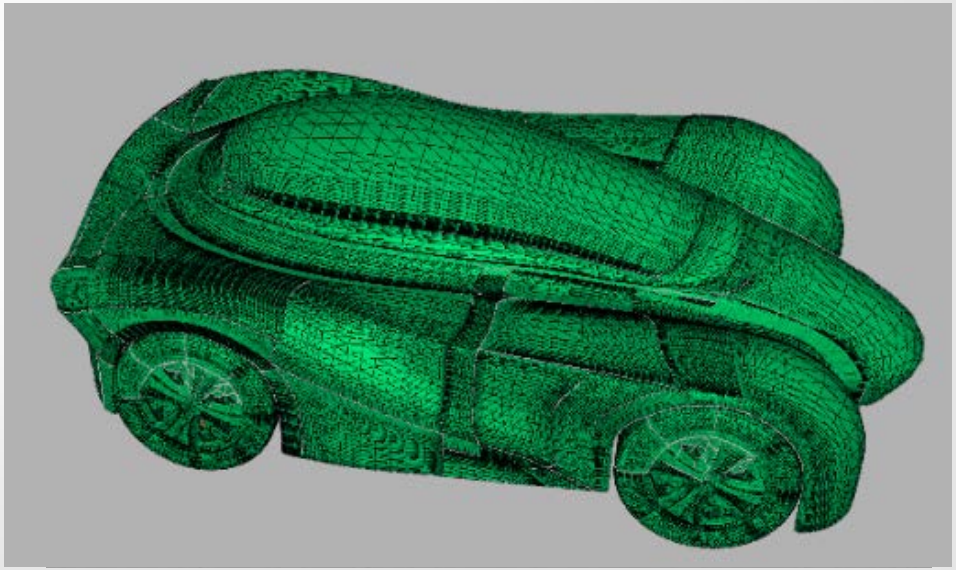






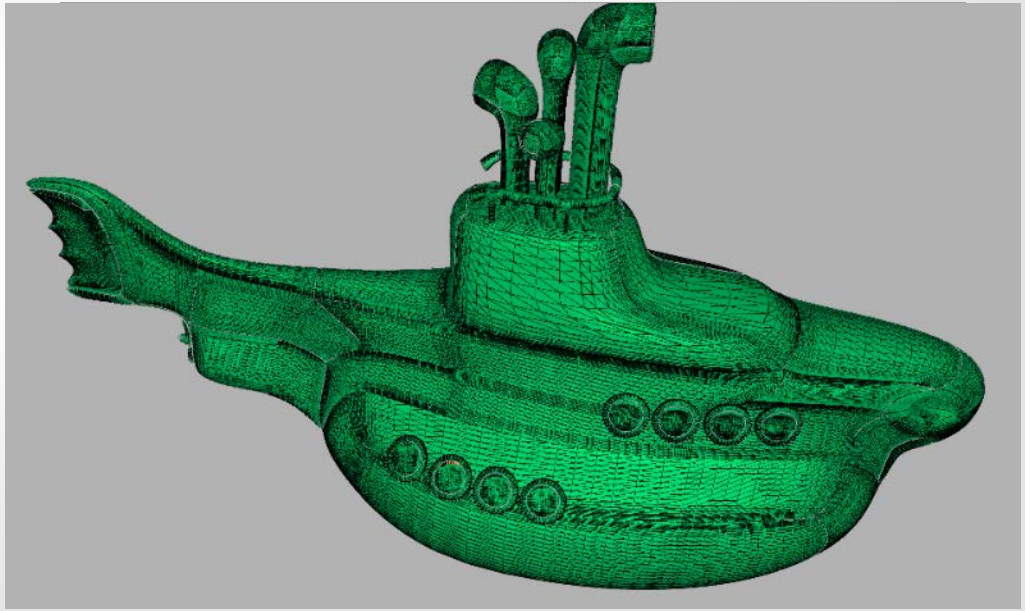
Results

- Car model
  - Error tolerance 0.05
  - 407 surfaces
  - 3,081,930 triangles
  - 10.4 ms



Results

- Submarine model
  - Error tolerance 0.05
  - 365 surfaces
  - 2,235,934 triangles
  - 9.16 ms



Acknowledgement

- This research is supported by Multi-plAtform Game Innovation Centre (MAGIC), funded by the Singapore National Research Foundation under its IDM Futures Funding Initiative and administered by the Interactive & Digital Media Programme Office, Media Development Authority of Singapore, and the National Science Foundation of China (Grant no. 60933008, 61272300).

GPU TECHNOLOGY CONFERENCE

Thank you!

NANYANG TECHNOLOGICAL UNIVERSITY

MAGIC
Multi-plAtform
Game Innovation Centre