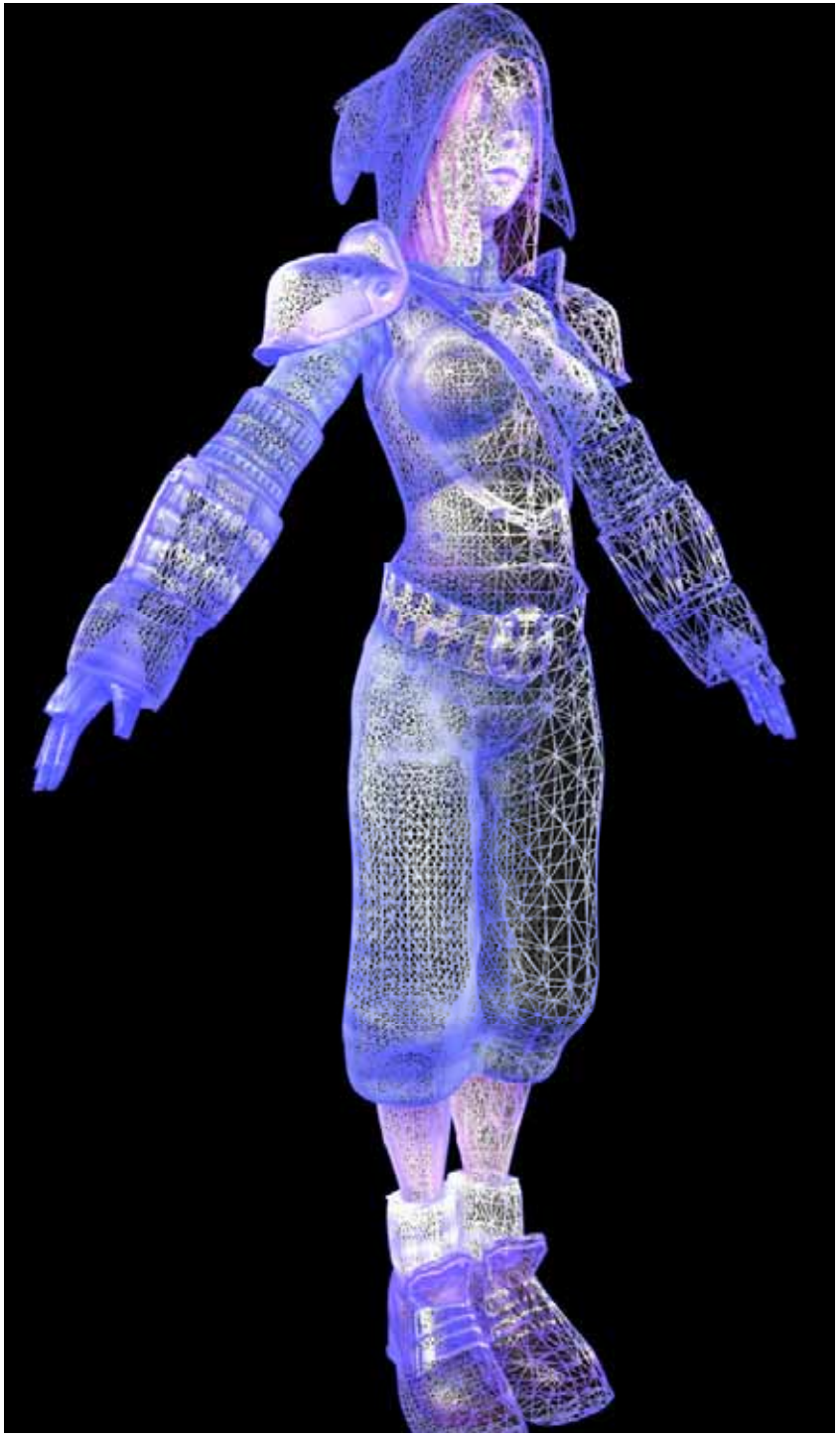
Instanced Tessellation in DirectX10

Andrei Tatarinov
NVIDIA Developer Technology
February 2008



Outline

- Motivation
- Tessellation basics
- Future tessellation model
- Instanced tessellation
- Computing tessellation factors
- Conclusions

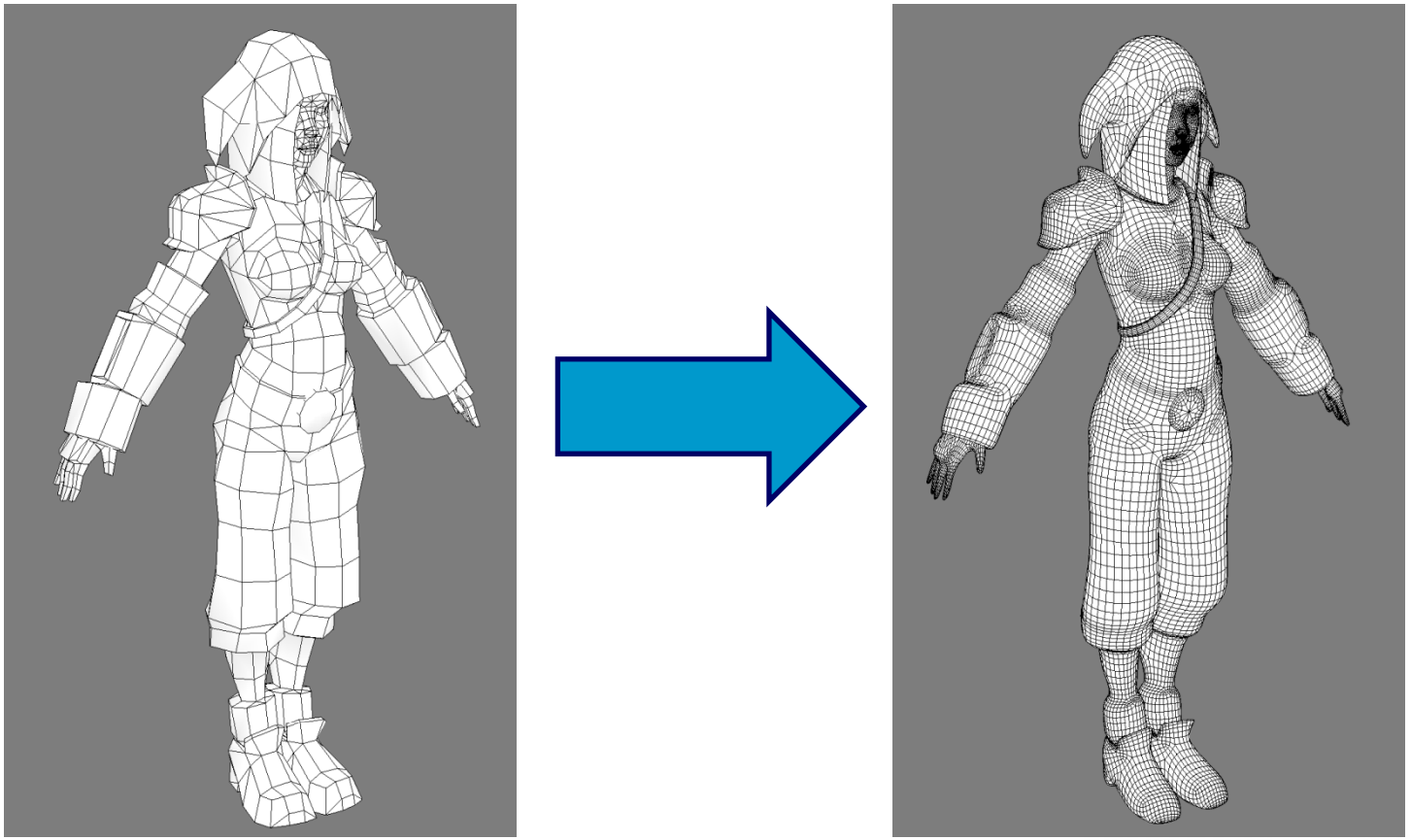
Motivation behind tessellation

Why do we need tessellation?

- To improve visual quality
  - Can add a lot of high-frequency detail
- To improve rendering performance
  - Saves memory and bandwidth
  - Lower computation frequency of animation/skinning
  - Natural scalability (LOD)

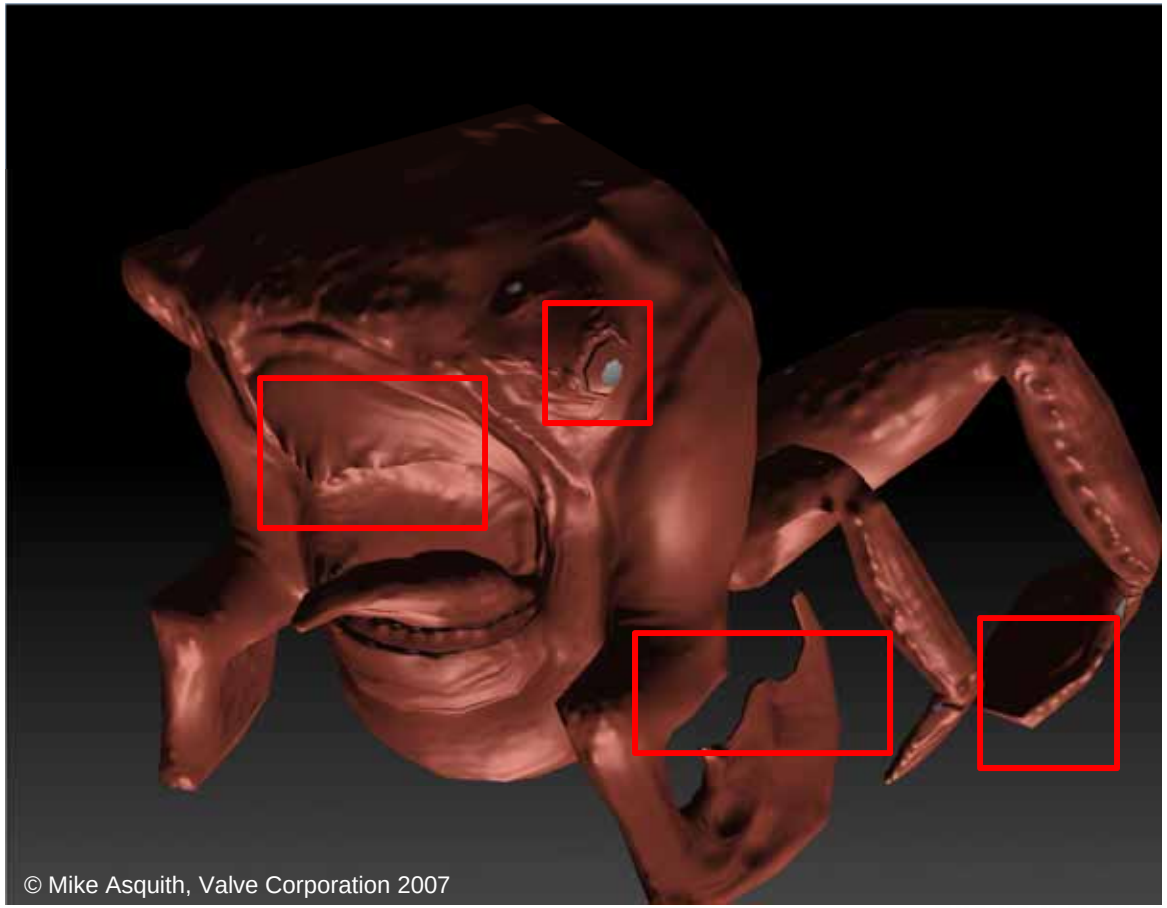
Increasing the number of primitives

- Makes models look more realistic



Increasing the number of primitives

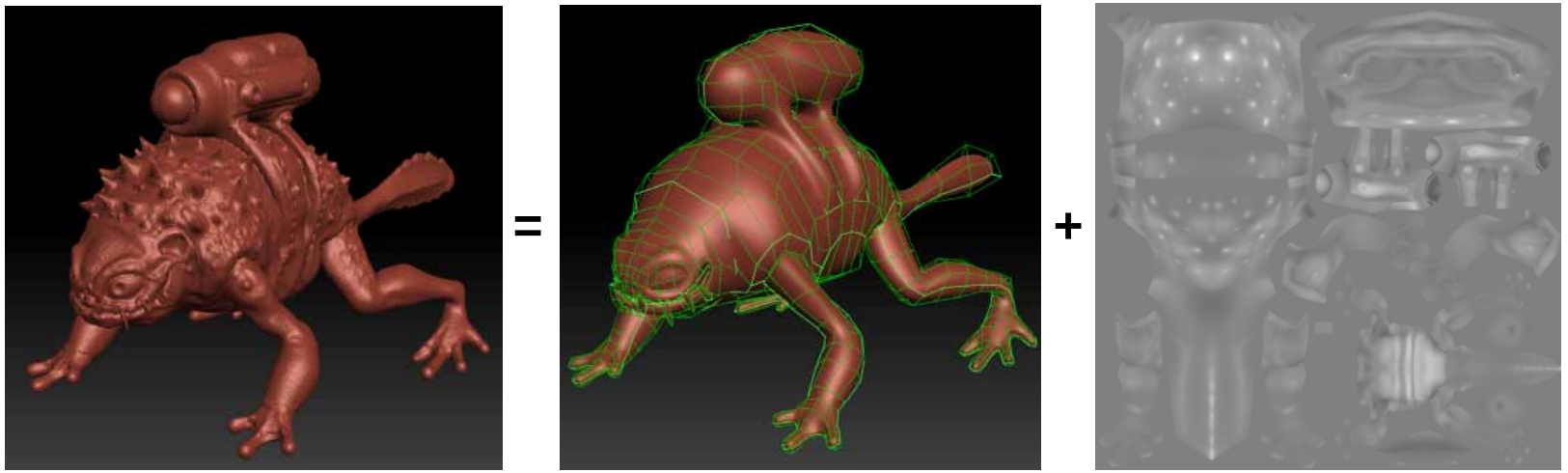
- Complex shading can't hide lack of detail



Increasing the number of primitives

Saving memory and bandwidth

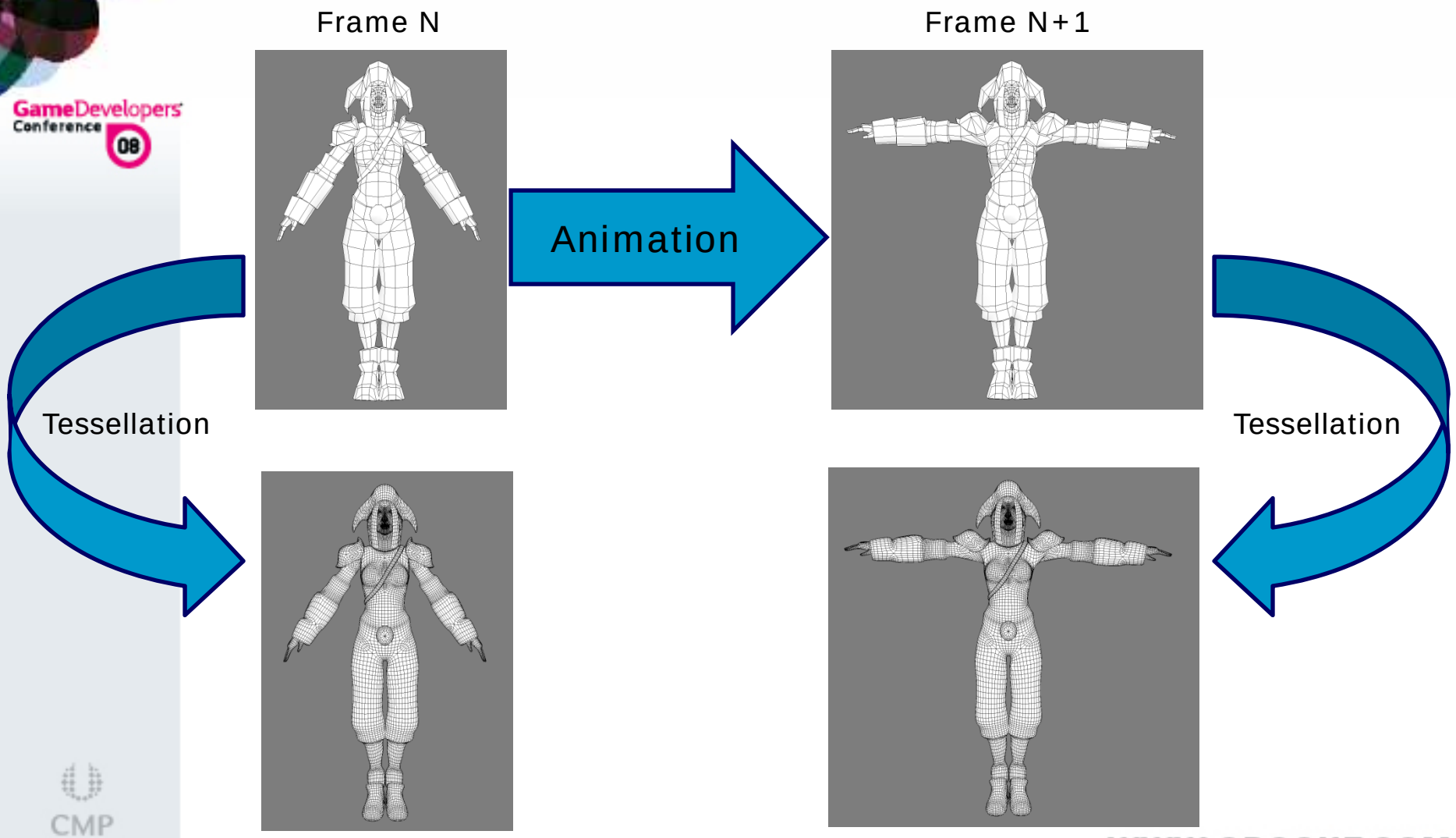
- High-detailed mesh can be represented with a coarse mesh and a displacement map
  - ALU performance scales faster than bandwidth



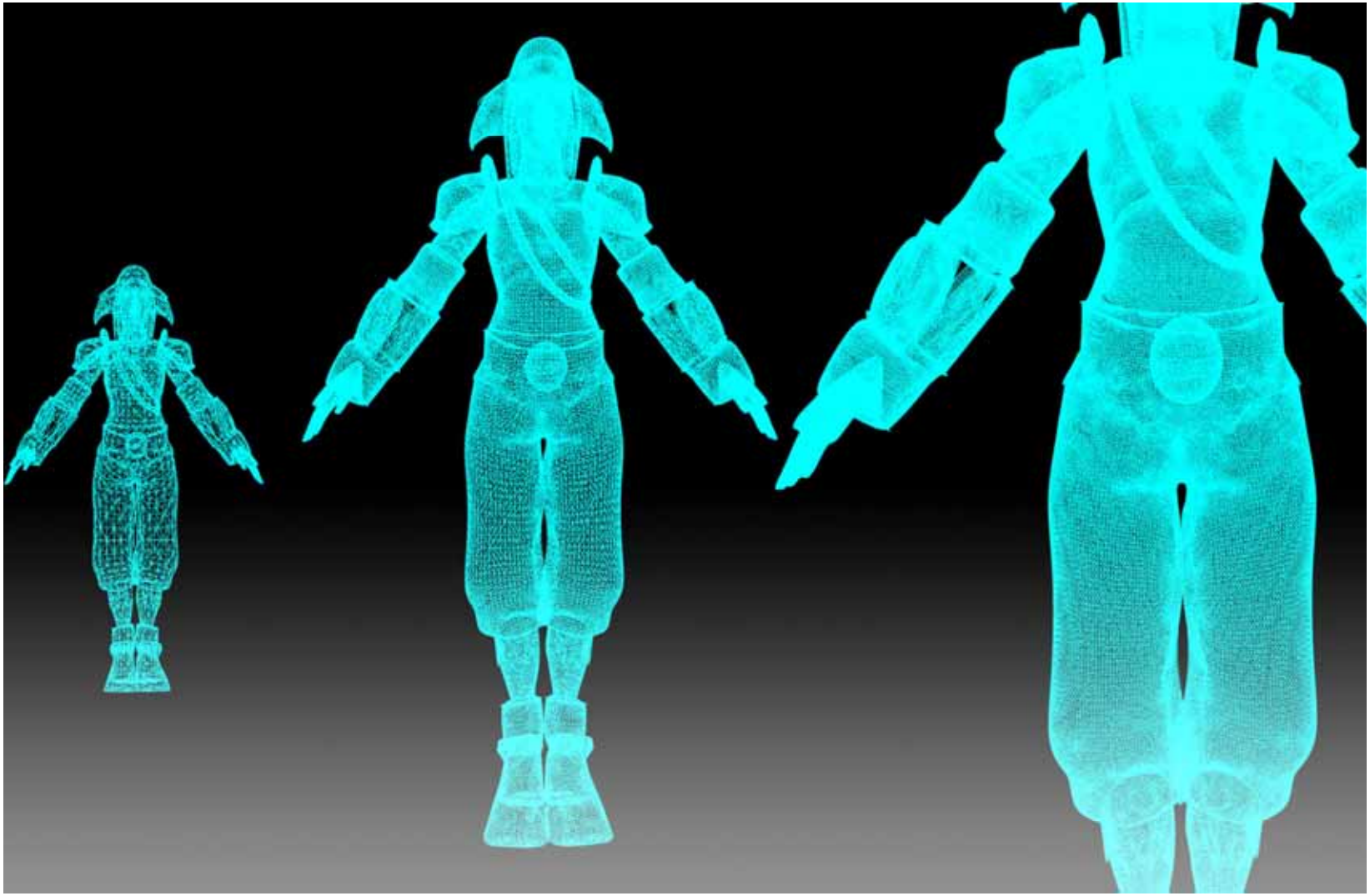
Animation

- Complex animation (morph targets, blend shapes) can be done on coarse rep

Animation



Scalability



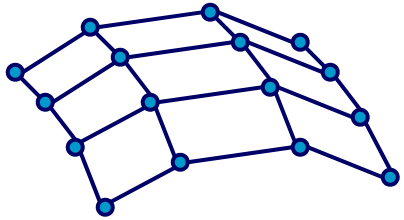
Conclusions

- Outstanding visual improvement
  - More details, more primitives
- Efficient rendering
  - Saves memory and bandwidth
  - Animation at lower rate
  - Scalability

Tessellation basics

Tessellation concept

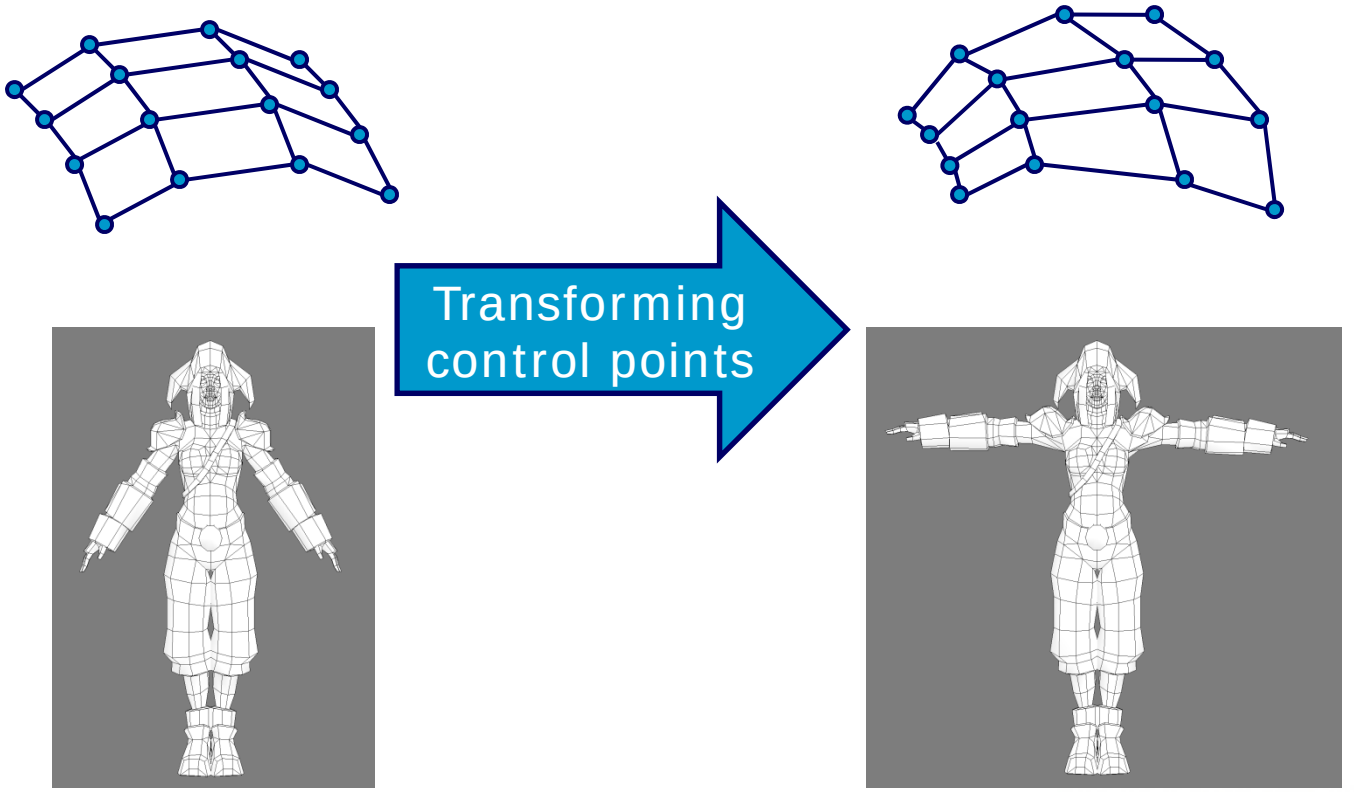
- A new primitive called **patch**
  - Patch is defined by a set of control points



- A new type of operation called **refinement**
  - Generate a number of triangles from a patch

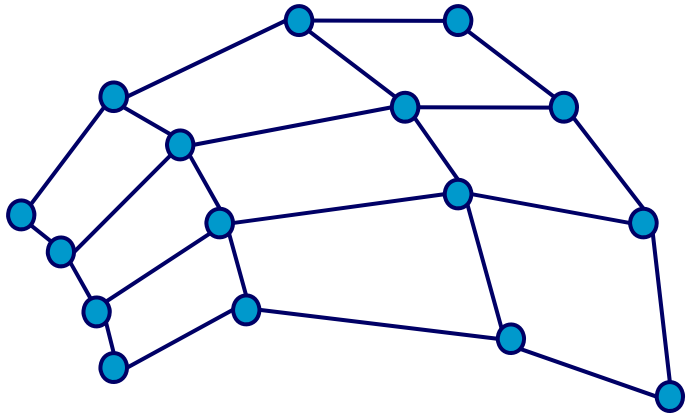
Transforming control points

- Animation can be performed at a lower rate



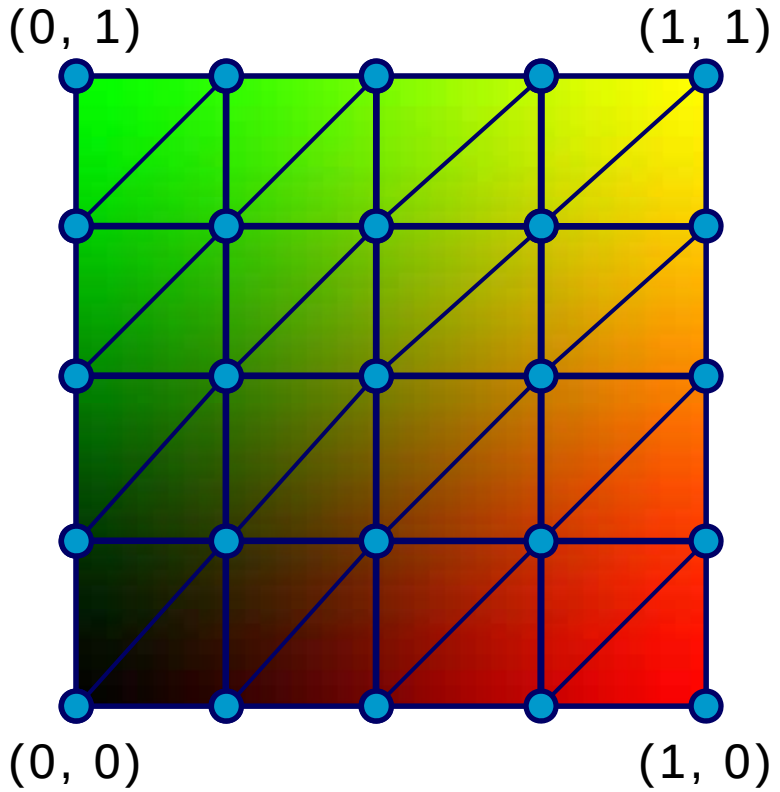
Per-patch operations

- LoD computation
- Transformation to another basis
  - Bezier -> B-spline
  - B-spline -> NURBS
  - etc.

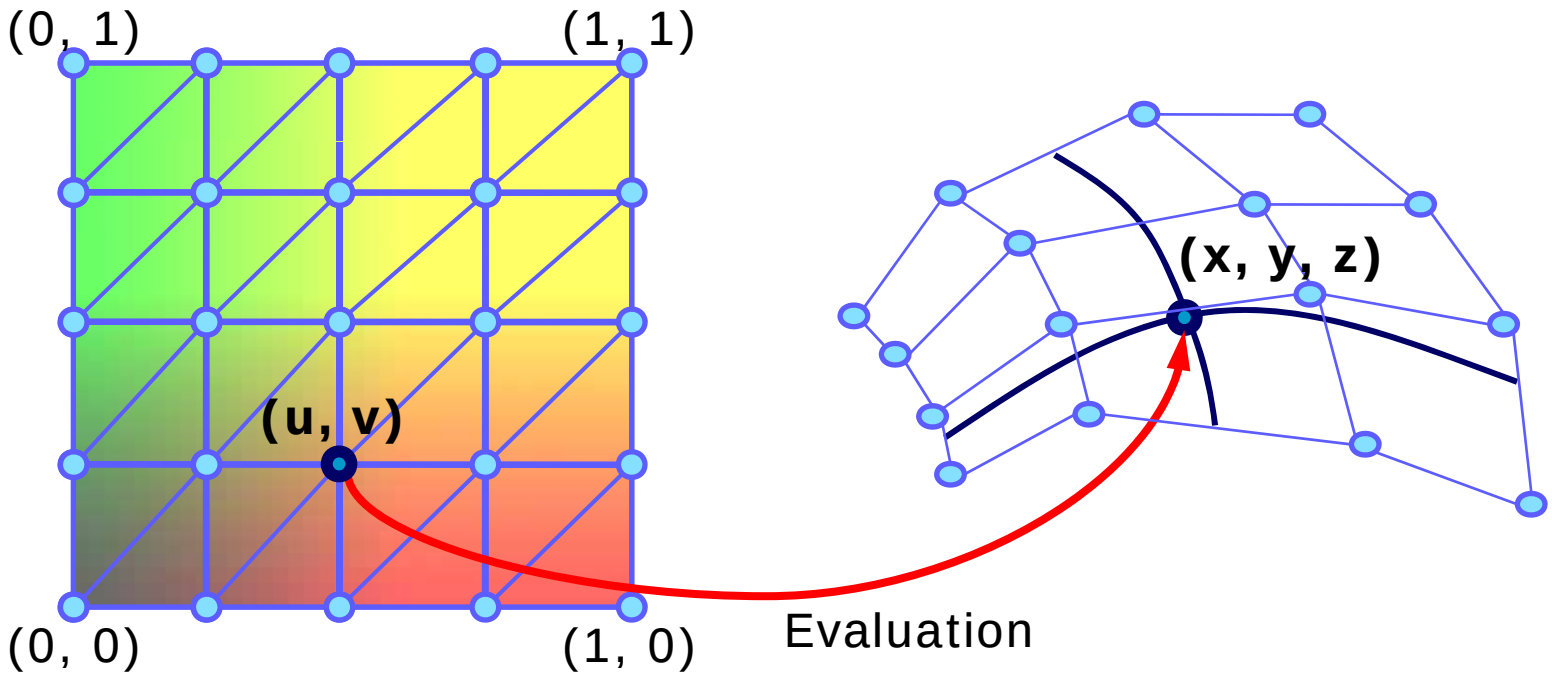


Generating topology

- Generate a set of *(u,v)*-points in the tessellation domain



Evaluating points



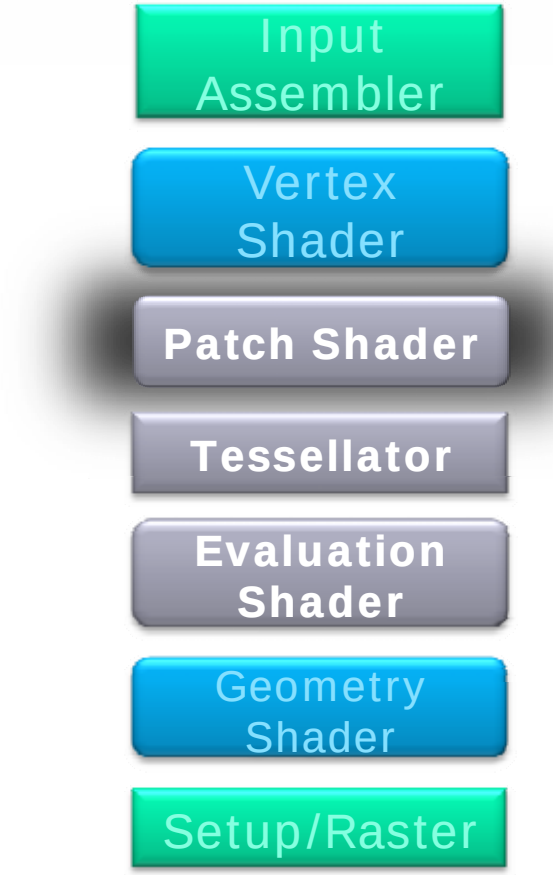
Future tessellation model

Tessellation pipeline

- New input primitive - **patch**
- Two new shader stages:
  - Patch Shader
  - Evaluation Shader
- One fixed function stage:
  - Tessellator

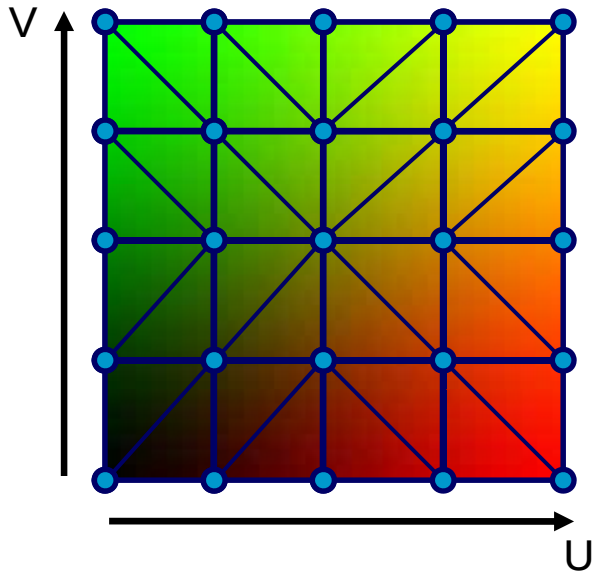
Patch shader

- Edge LODs computations
- Basis conversion for easier evaluation



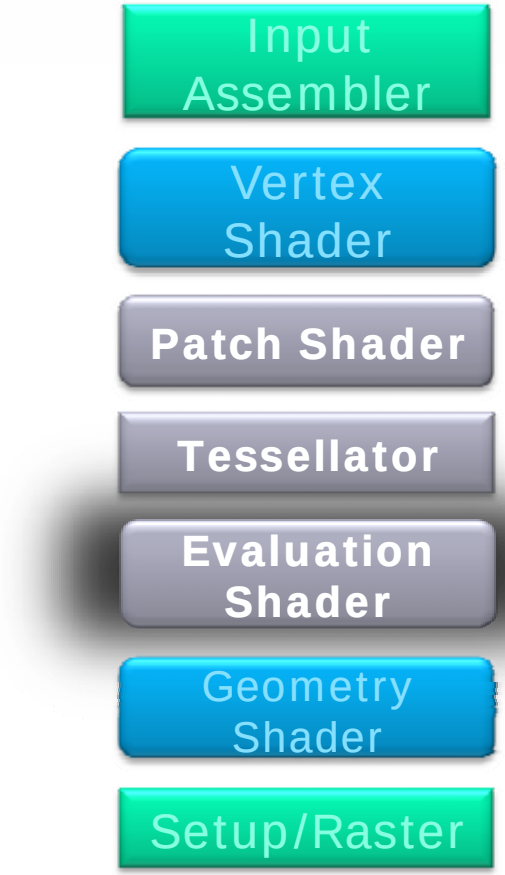
Tessellator

- Fixed function stage, but configurable
- Generates UV coordinates in the tessellation domain



Evaluation shader

- Evaluates surface given parametric UV coordinates
  - Sees all control points for a patch
- Applies displacement mapping, space transformations, etc.



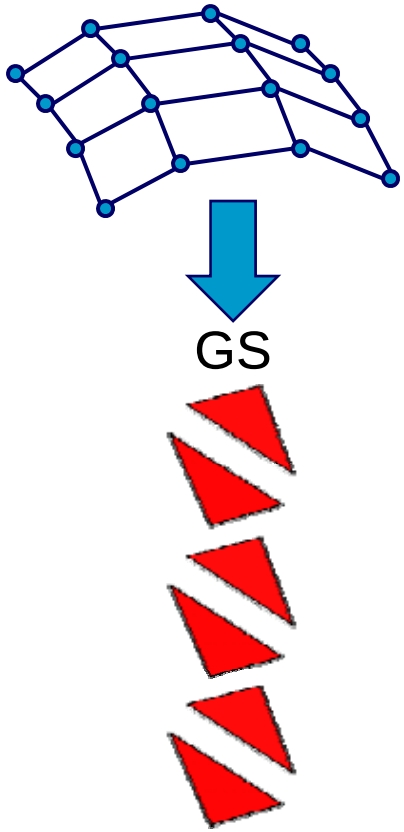
Instanced tessellation

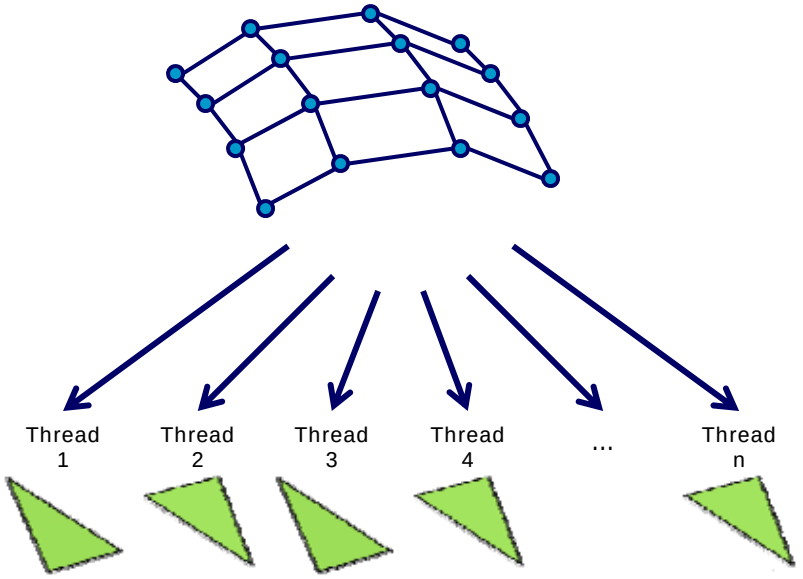
Next Gen content today!

- We can approximate the tessellation pipeline with **DirectX 10 API**
- Instancing can be used to replicate patches

GS is not for tessellation

- GS outputs triangles serially



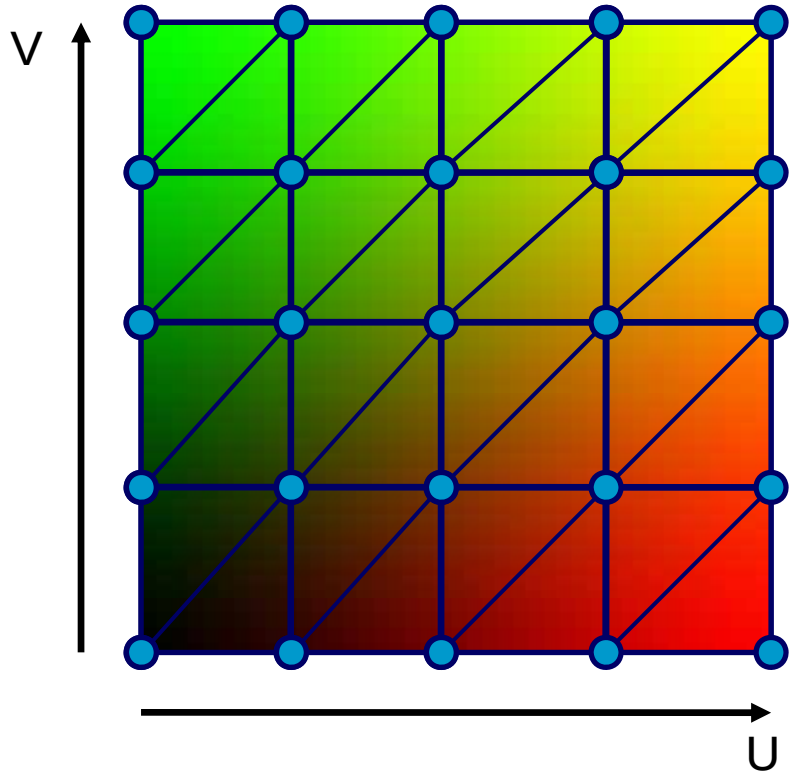


GS is not for tessellation

- Limited output size (maximum 1024 scalars) is not always enough
  - If each vertex is 4 float's, you can only tessellate up to 16x16

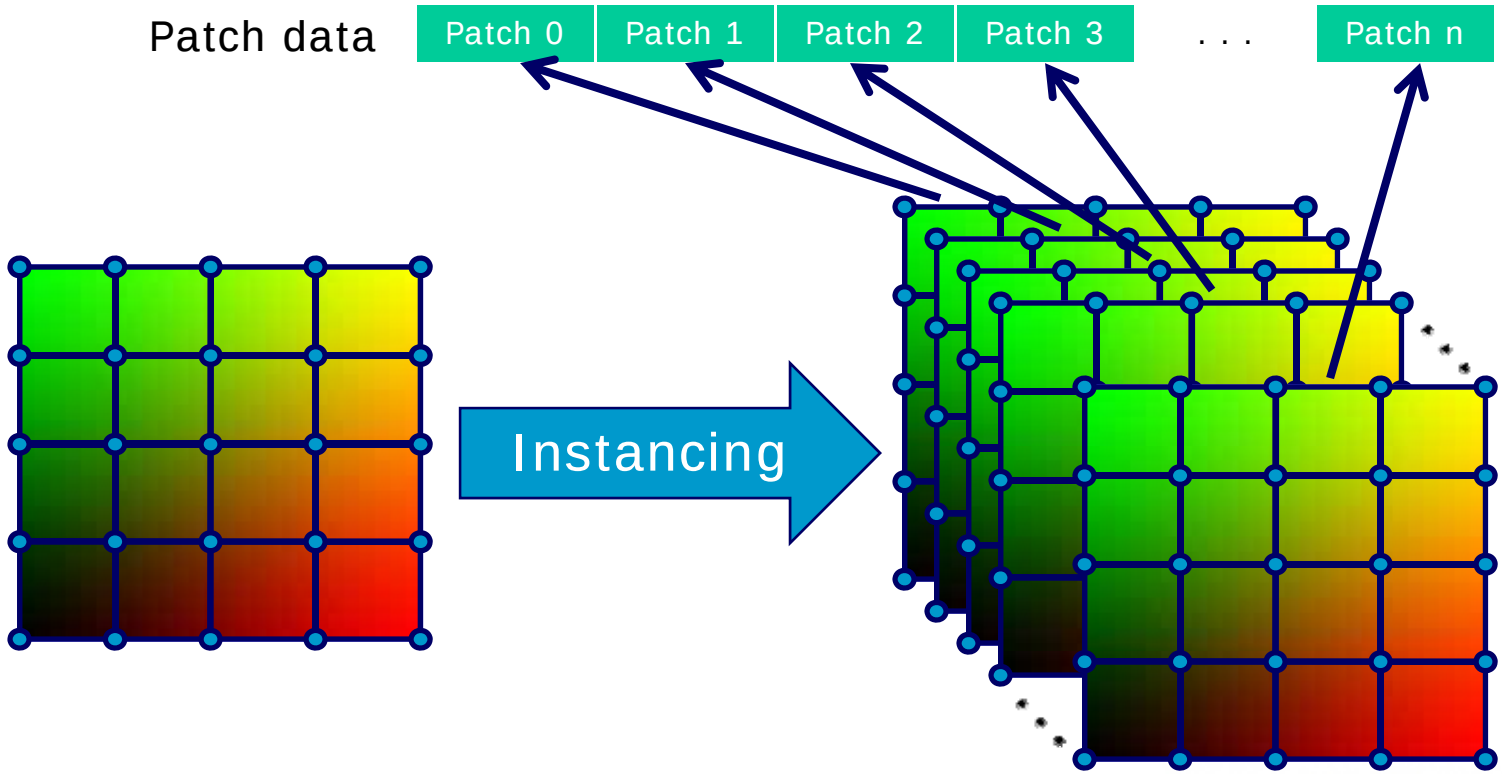
Use instancing instead

- Render pretessellated patch with instancing
- Set the entire mesh as instance data



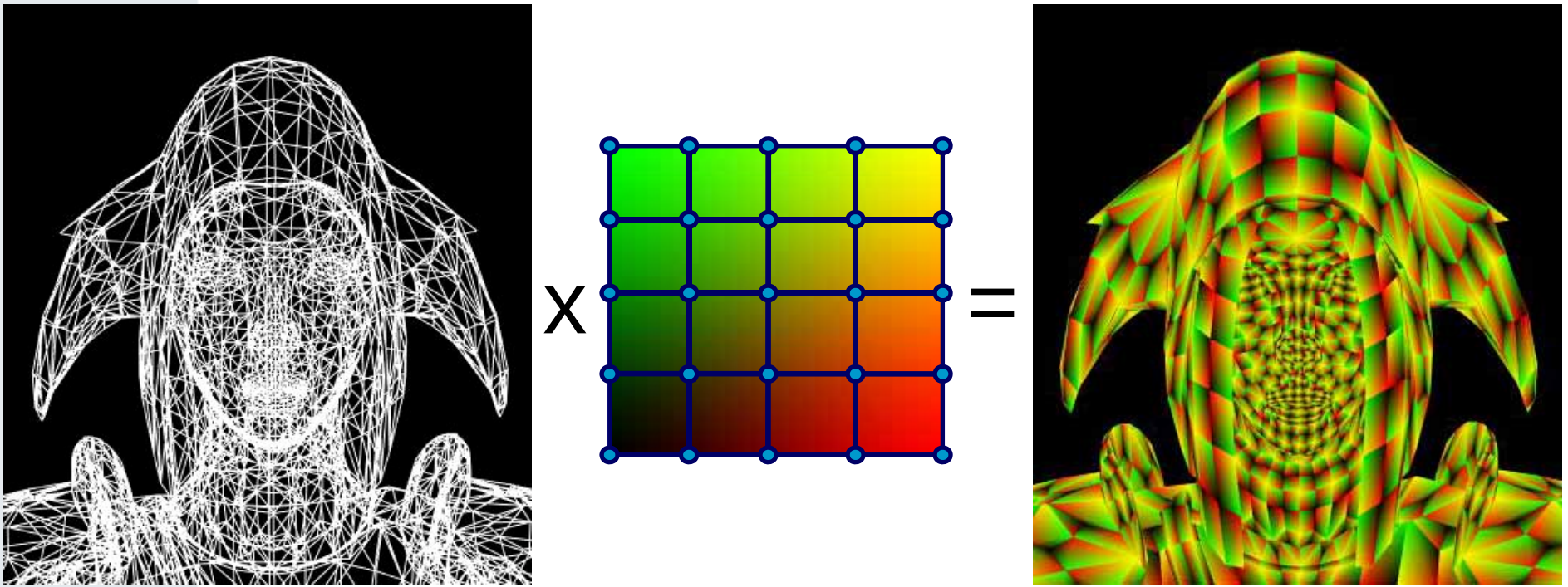
Use instancing instead

- Render pretessellated patch with instance count equal to patch count in the mesh



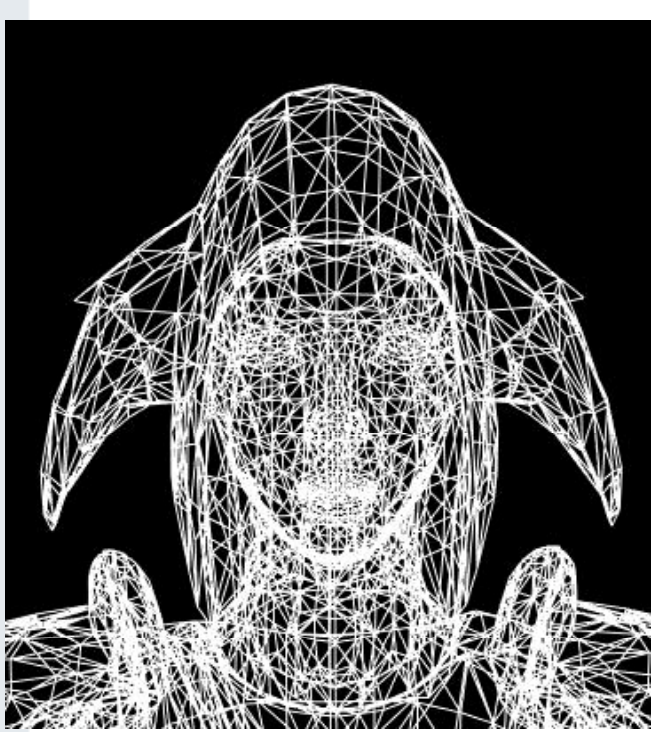
Use instancing instead

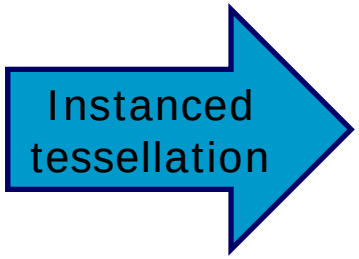
- Pretessellated patch represents results of tessellating every input patch

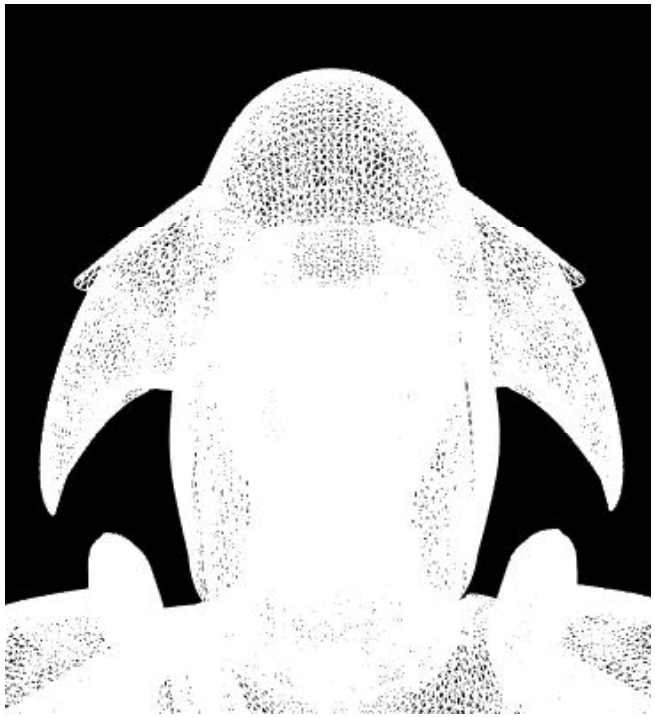


Use instancing instead

- Compute refined vertex position in the **vertex shader** using chosen evaluation algorithm





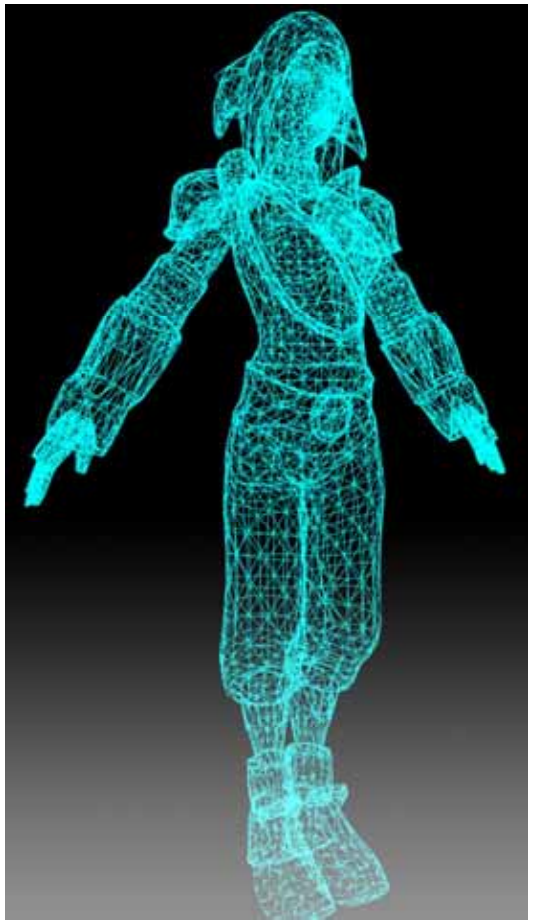


Input attributes limitation

- Maximum VS input size is not enough to fit all data required for point evaluation
- Instead all data can be stored in buffers bound as shader resources
  - Use Load() instruction to fetch this data

Using Load()

- Store mesh data in vertex buffers
- Bind these buffers as shader resources



Control points

Patch 0	Patch 1	Patch 2	Patch 3	. . .	Patch n

Tangents

Patch 0	Patch 1	Patch 2	Patch 3	. . .	Patch n

Bitangents

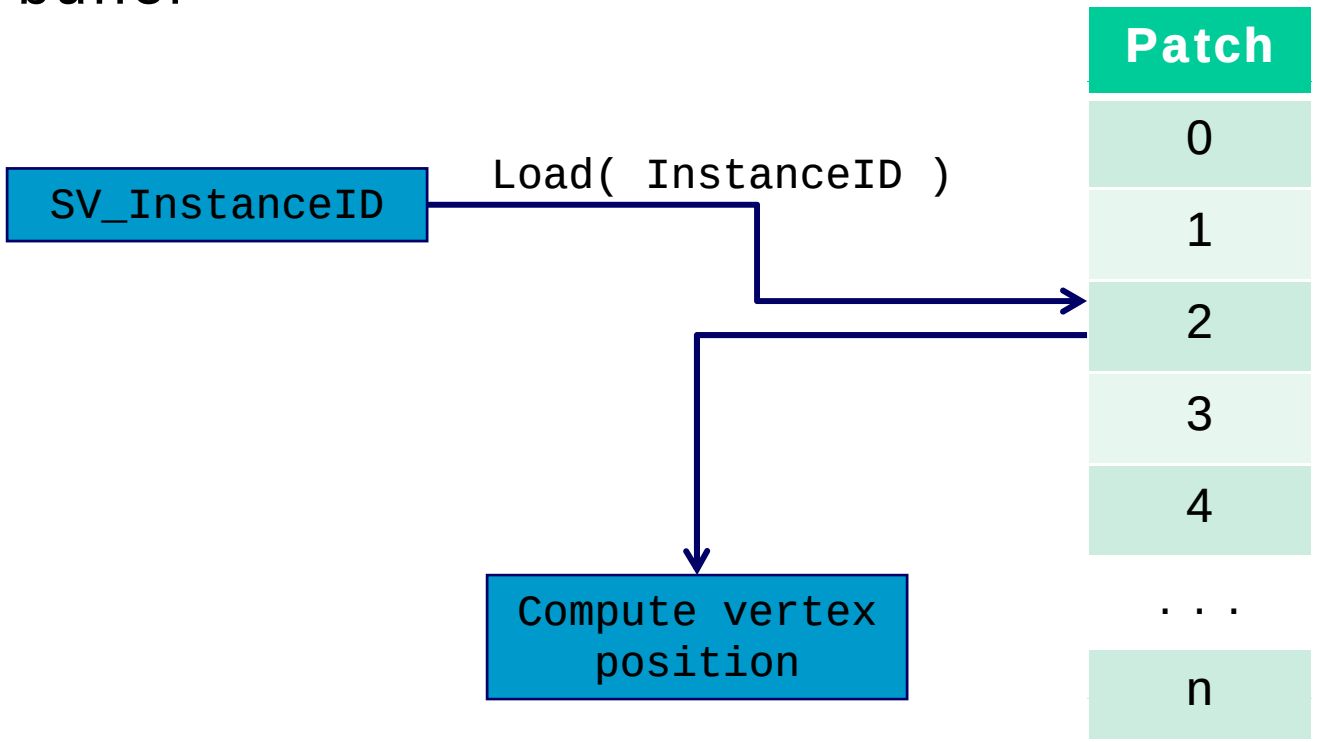
Patch 0	Patch 1	Patch 2	Patch 3	. . .	Patch n

Texture coordinates

Patch 0	Patch 1	Patch 2	Patch 3	. . .	Patch n

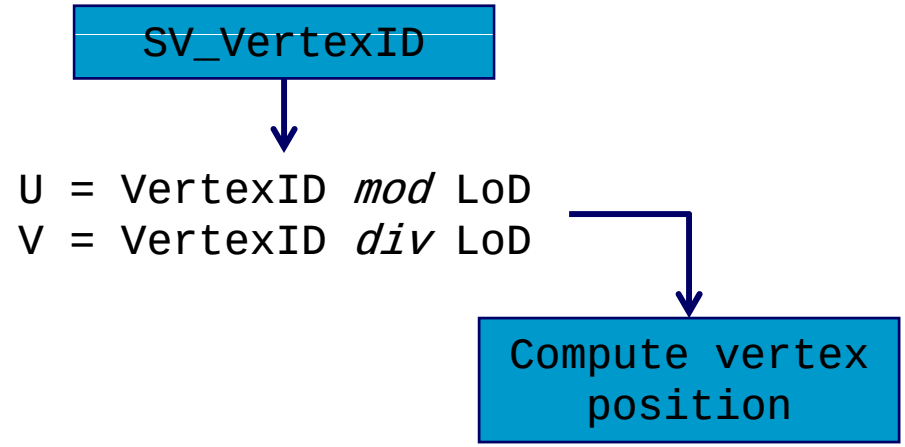
Load() and InstanceID

- Use SV_InstanceID as an index to the patch buffer



Computing U and V coords

- Use SV_VertexID to compute U and V coordinates for the current vertex



- SV_VertexID and SV_InstanceID is the only VS input

Application integration

- Without tessellation

```
Bind_mesh_vertex_buffer();

Bind_VS();
Bind_PS();

Draw ( primitives_count );
```

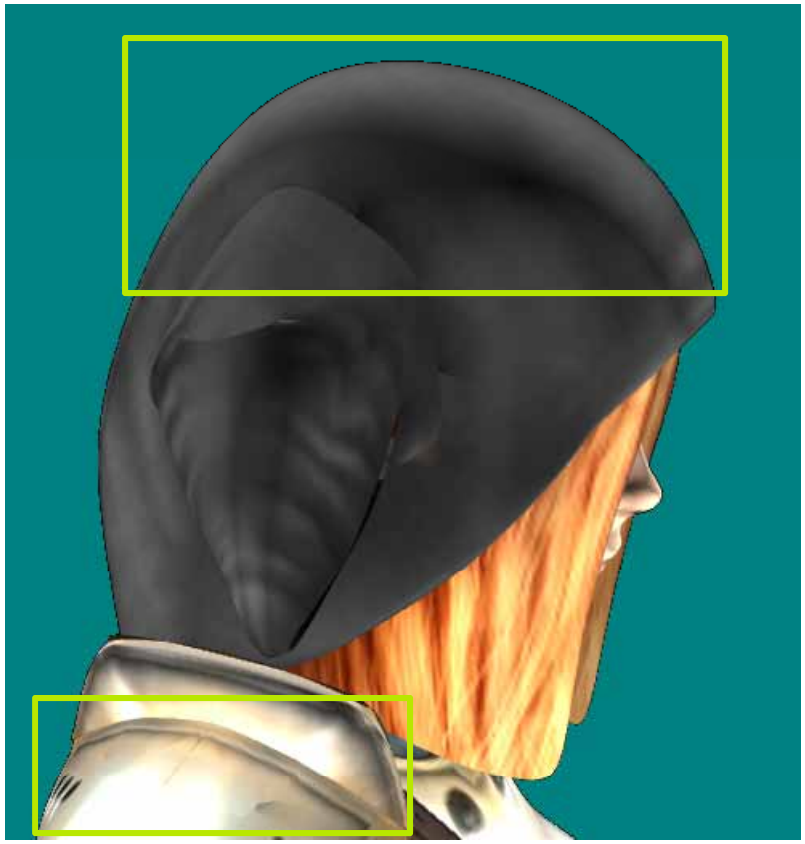
- With tessellation

```
Bind_mesh_vertex_buffer_SRV();

Bind_tessellation_VS();
Bind_PS();

Draw_instanced
    ( primitives_count,
      refined_vertex_count );
```

Results of refinement

- Before

- After



Results of refinement

- Before

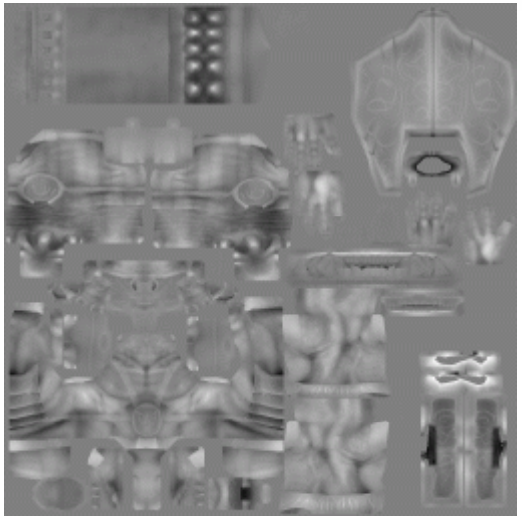
- After

Tips

- Pack all data in float4-buffers to use Load() more efficiently
- Use 2^n x 2^n tessellation and bitwise operations to compute U and V from Vertex_ID
  - Integer division is slow!

Adding displacement

- Can add true per-vertex displacement for the refined mesh



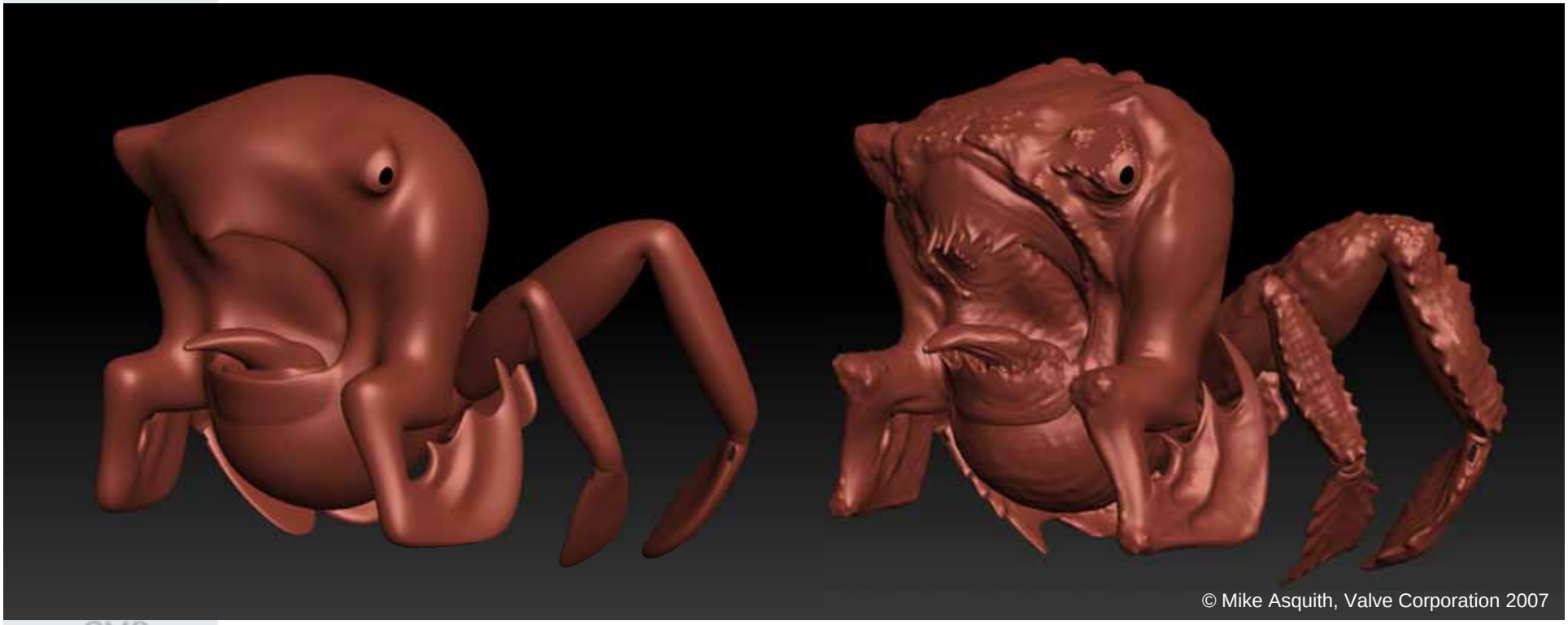
Adding displacement

- Use vertex texture fetch from a height map to add higher frequency details
  - Diffuse map

  - Height map

Adding displacement

- Original model
- Displaced model



Computing tessellation factors

Why do we need LoD?

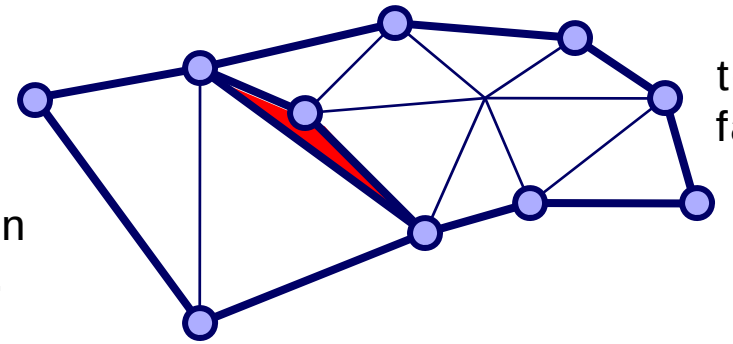
- Tessellation naturally supports dynamic LoD
- Allows to scale the number of primitives with distance or object size
- Makes rendering more efficient

LoD and instancing

- Patches can use different tessellation factors
  - To match triangle sizes to pixels on a screen
  - To prevent aliasing
- This is a problem, since in instancing we can use only one mesh which represents one particular tessellation factor

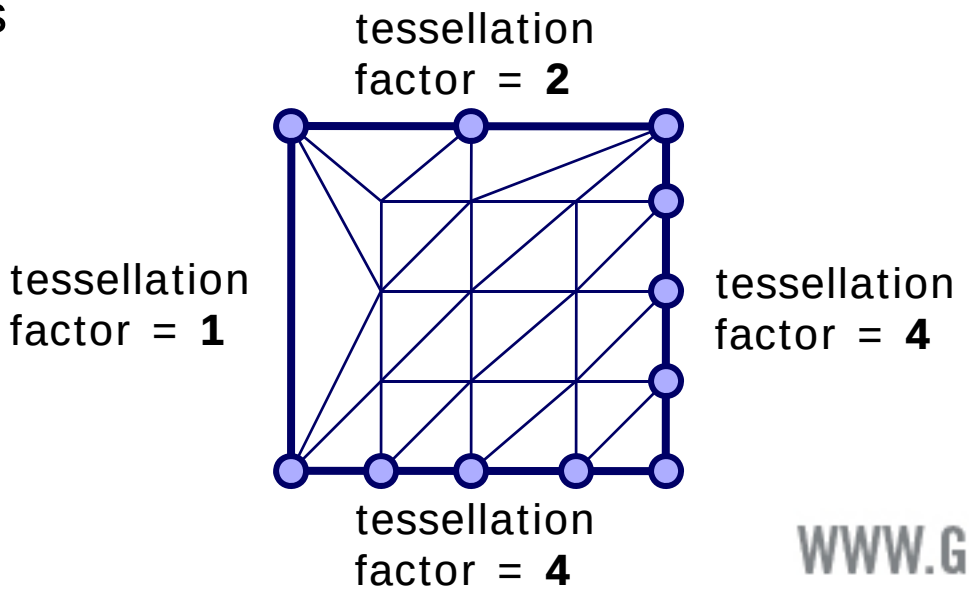
Computing tessellation factors

- Adjacent patches must agree on tessellation factor
  - Otherwise cracks can appear



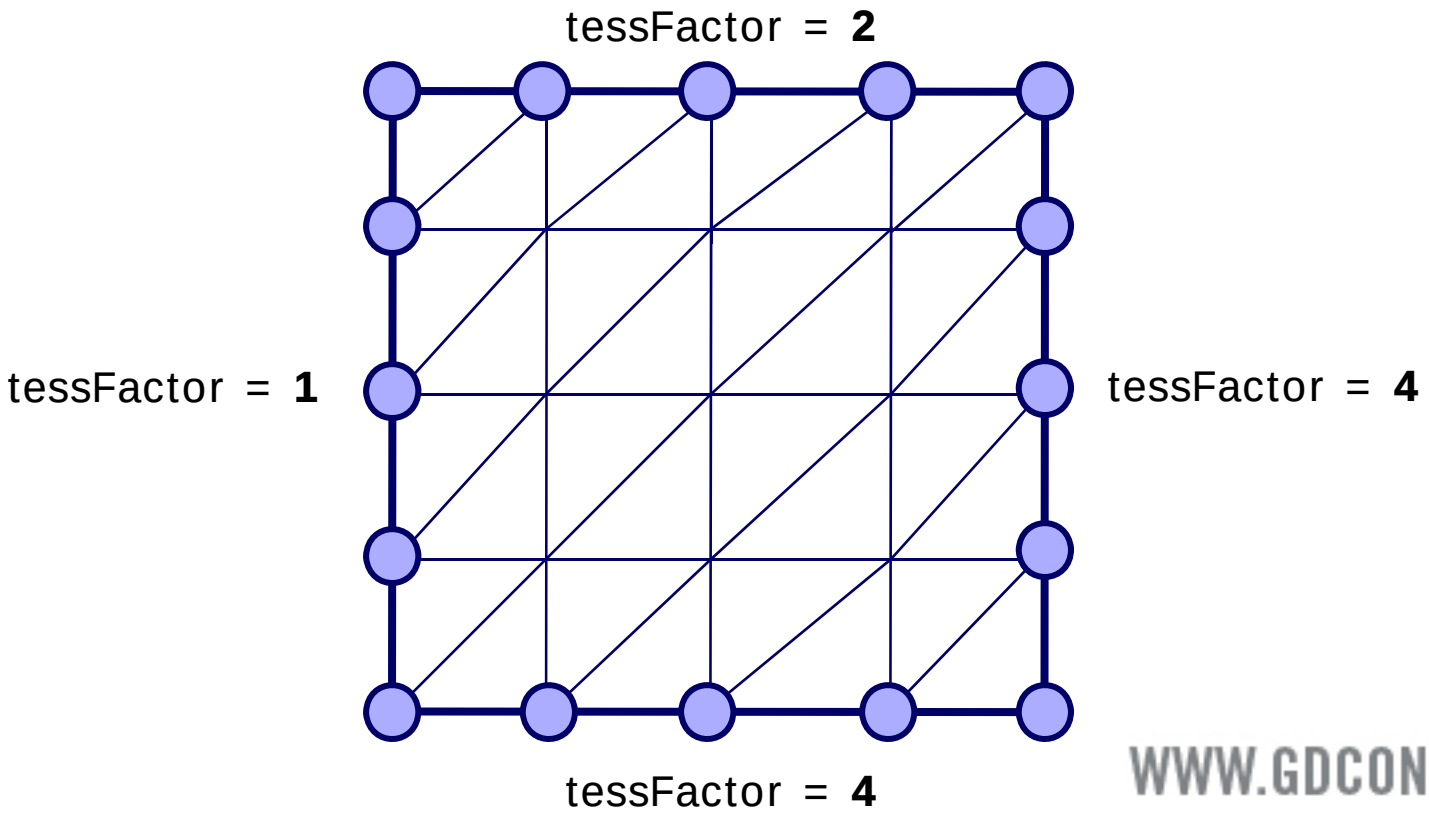
Computing tessellation factors

- We can compute tessellation factors per edge
  - Adjacent patches will have the same factors at shared edges
  - Patch tessellation factor is maximum of it's edge factors



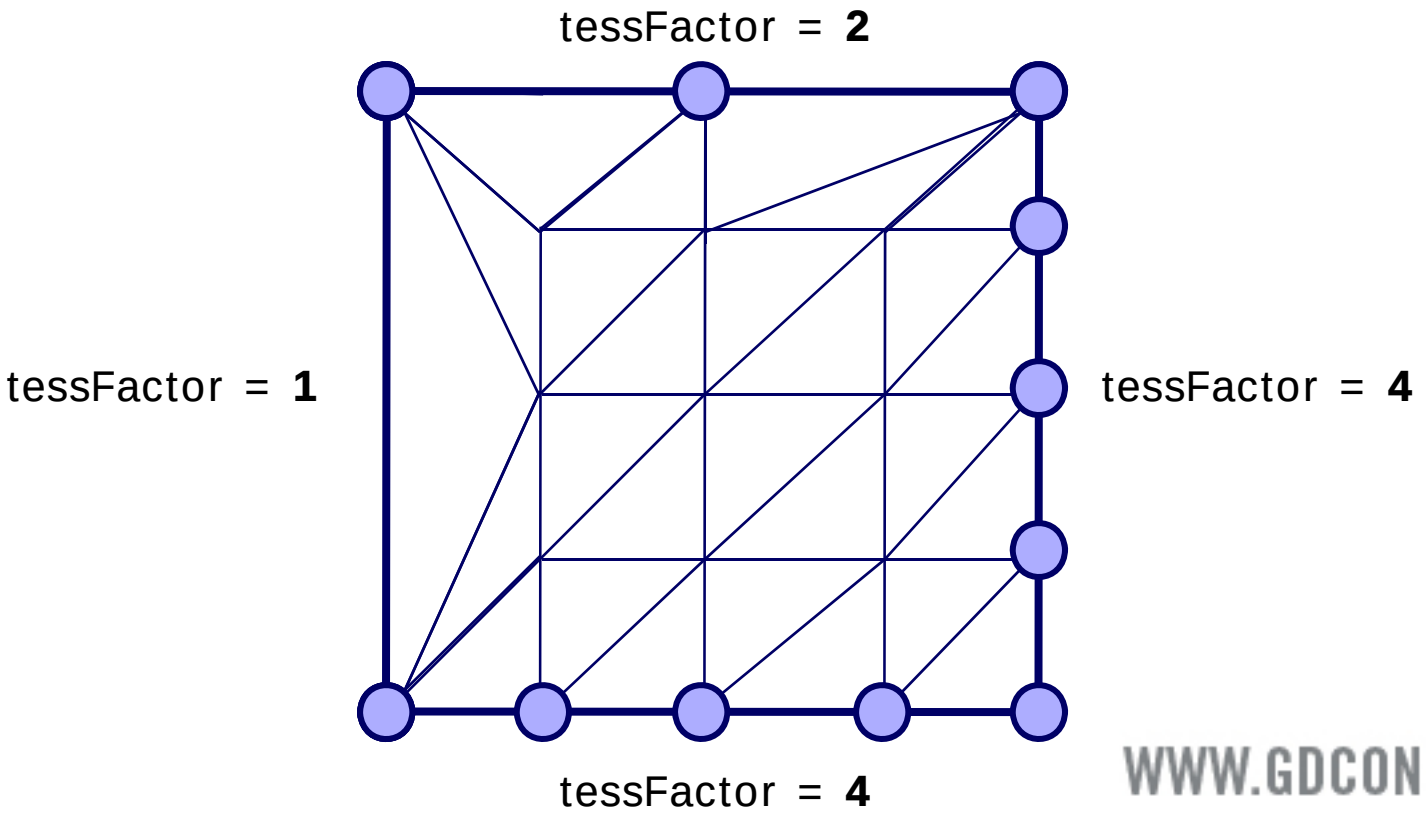
Computing tessellation factors

- Vertices can be moved to accommodate new tessellation factors



Computing tessellation factors

- Vertices can be moved to accommodate new tessellation factors

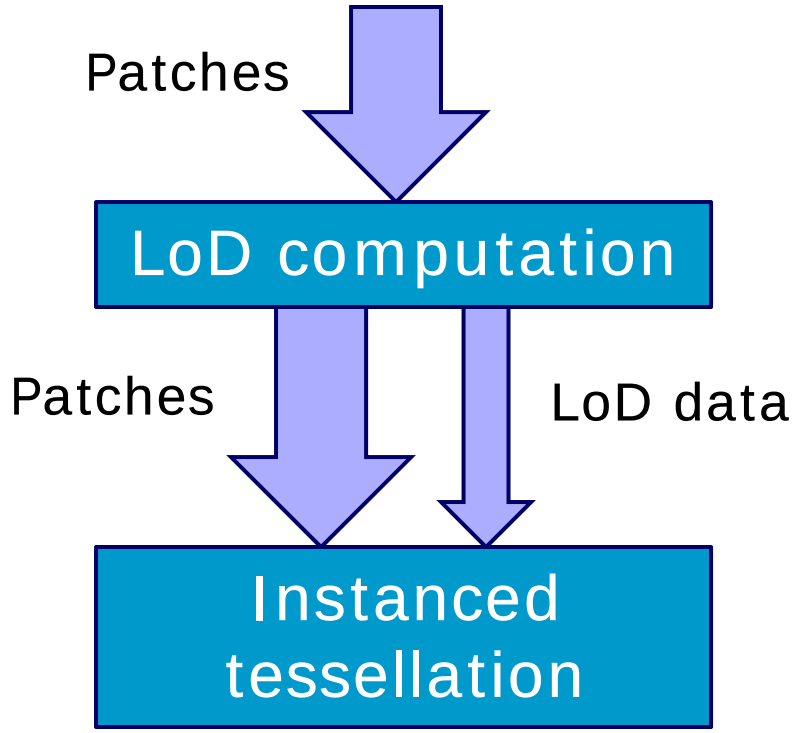


Computing tessellation factors

- Use different metrics for tessellation factors
  - Distance
  - Screen projection size
  - Curvature

Computing tessellation factors

- Add an extra stream-out pass which outputs four factor values per patch

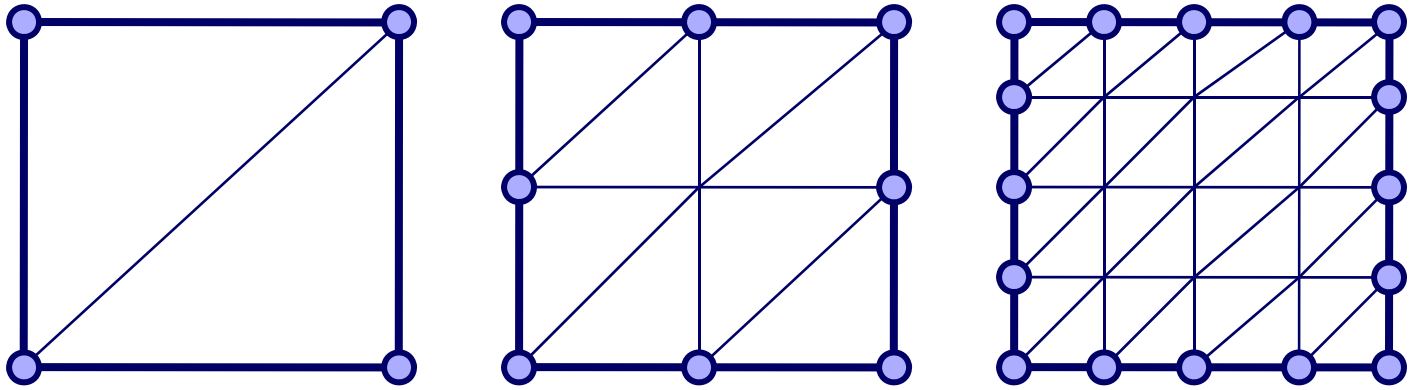


Tessellation factors usage

- Per-object factor – one tessellation factor applied to the whole object
- Per-patch factor – tessellation factors change across the mesh

Per-object factor

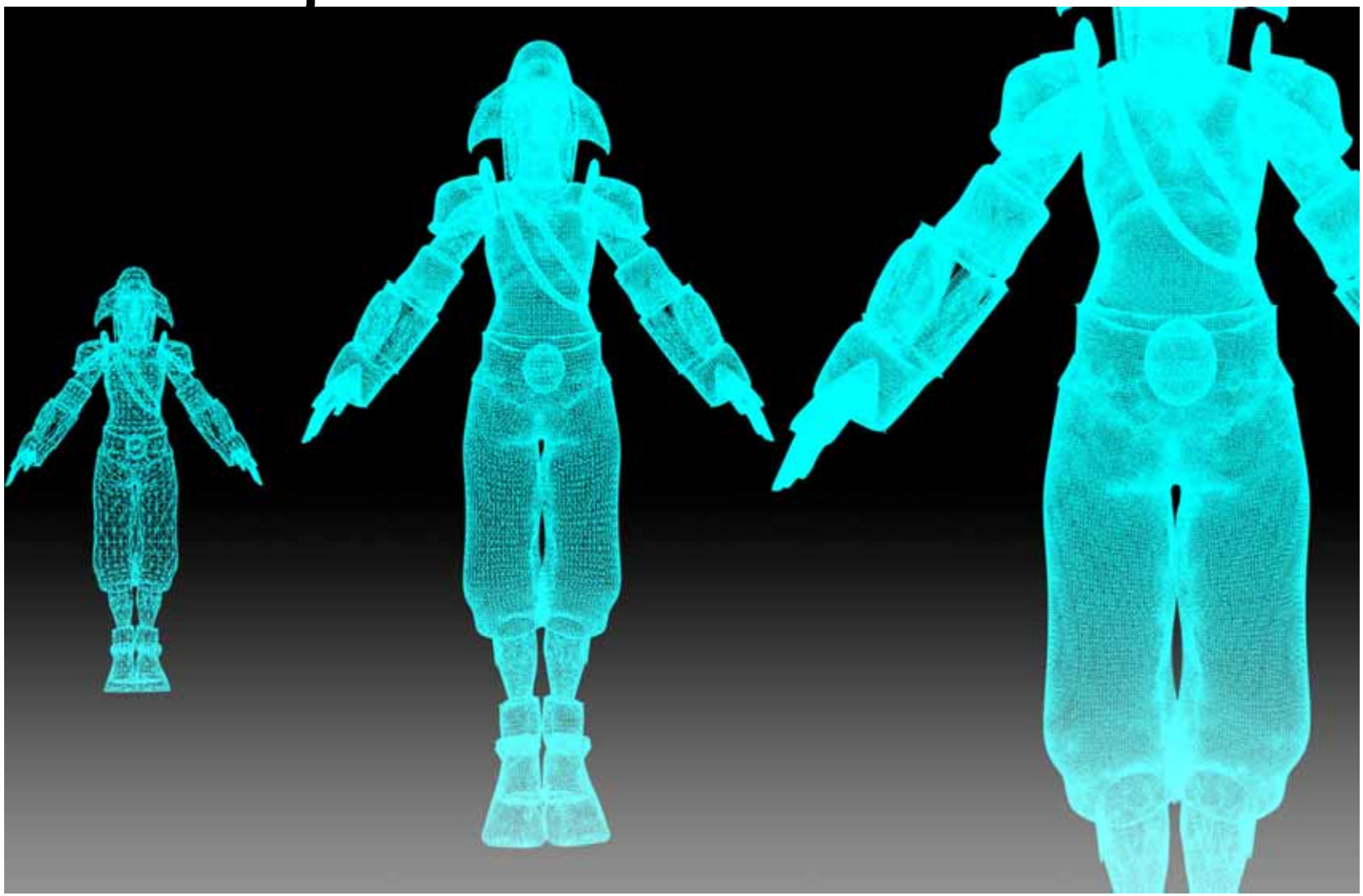
- Useful when applied to small objects and characters
- Use a set of pretessellated patches with different factors



Factor = 1 Factor = 2 Factor = 4

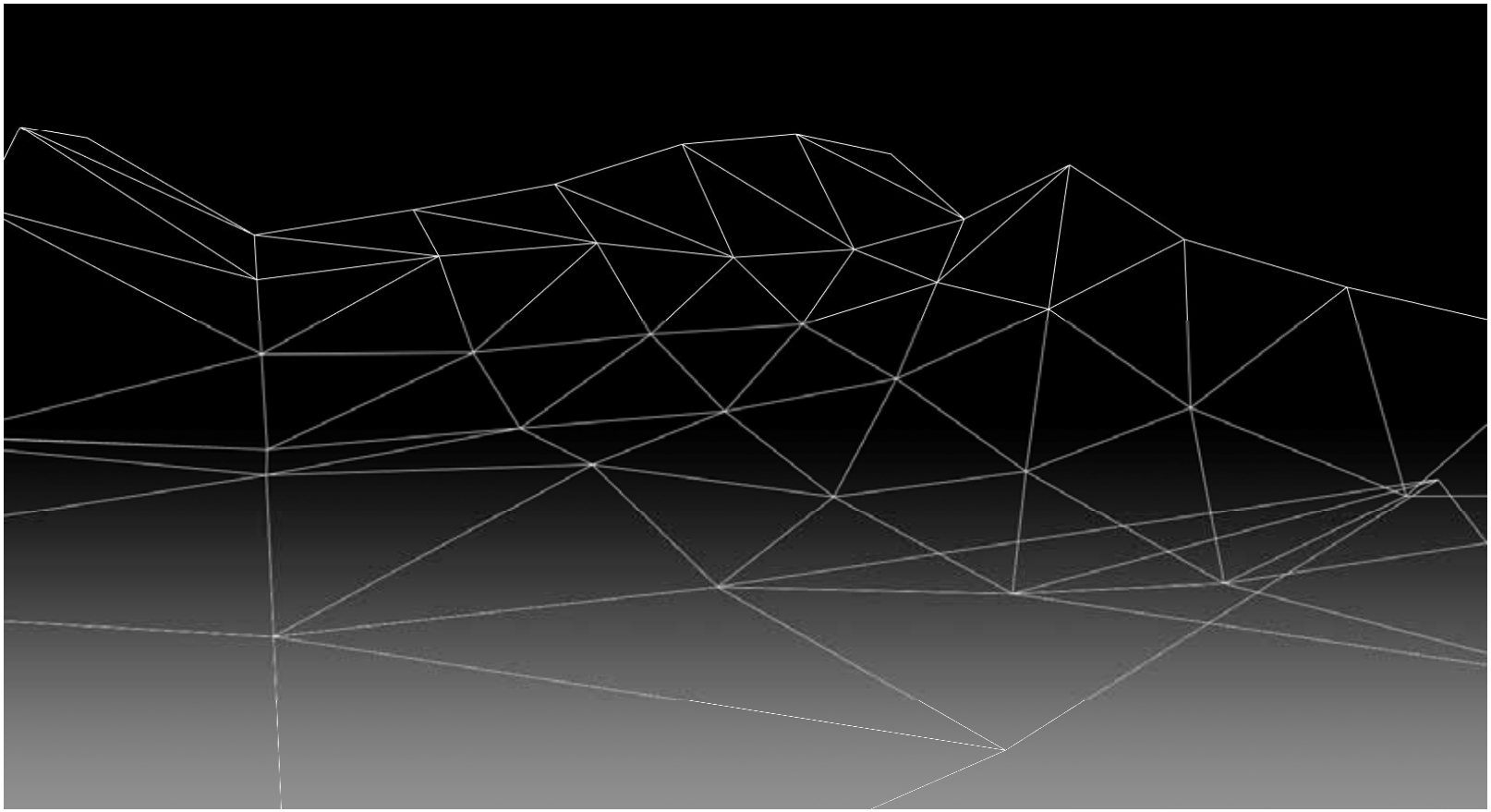
- Use a selected metric to find a tessellation factor and the appropriate patch

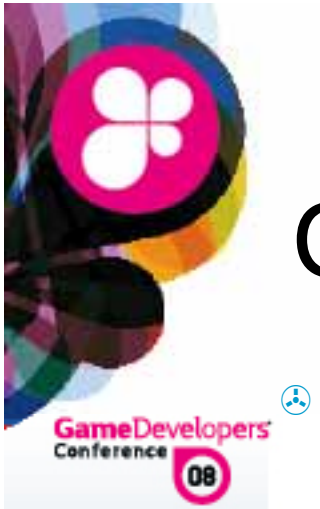
Per-object factor



Per-patch factor

- Can't use per-object factors for large meshes, such as landscapes





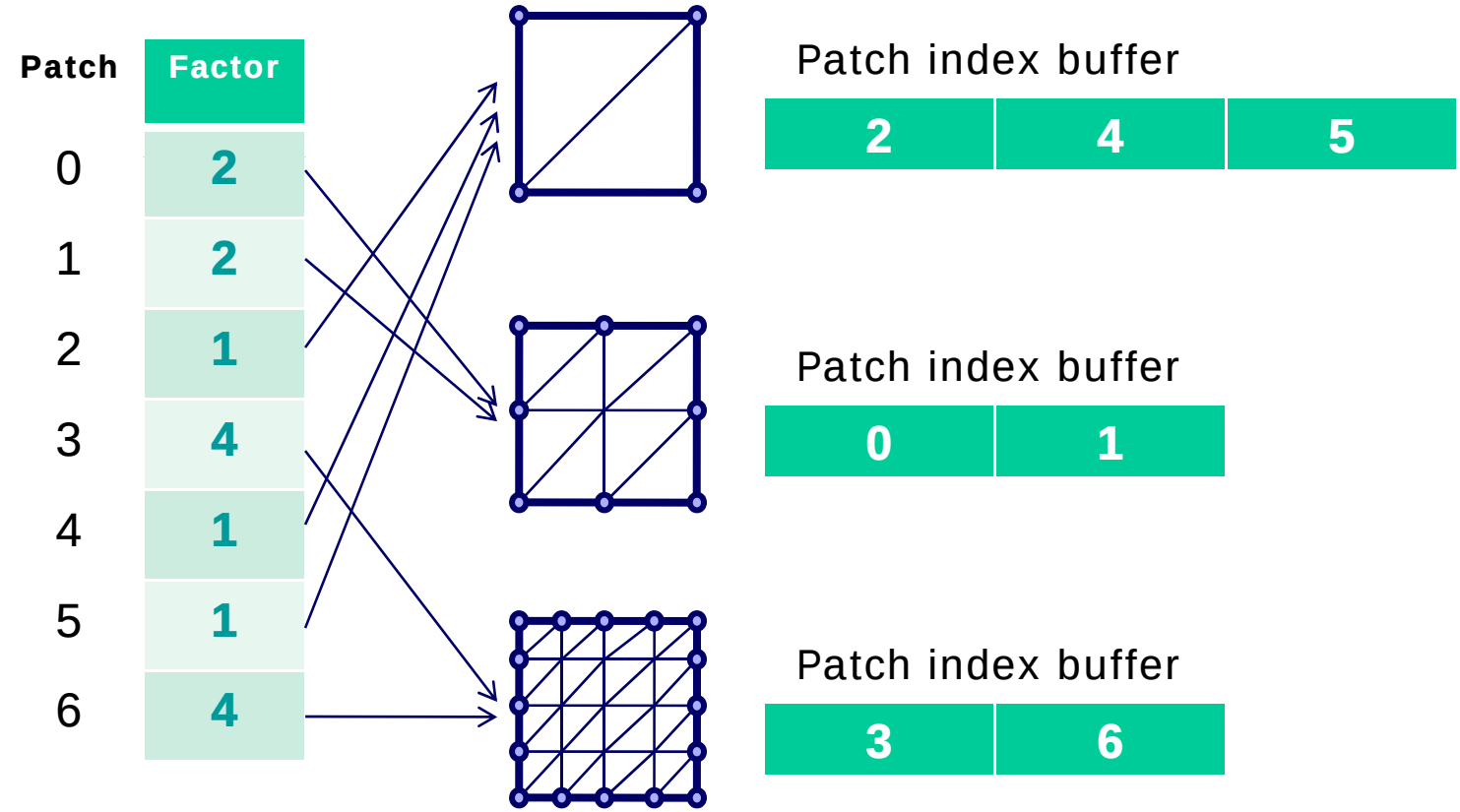
Computing patch factors

- Patch tessellation factor is a maximum of edge factors

Edge factors					Patch factor
1	2	2	2		2
2	2	1	2		2
1	1	1	1		1
4	**4**	**2**	**1**	`Max( edgeFactors )` →	**4**
1	1	1	1		1
1	1	1	1		1
2	4	4	2		4

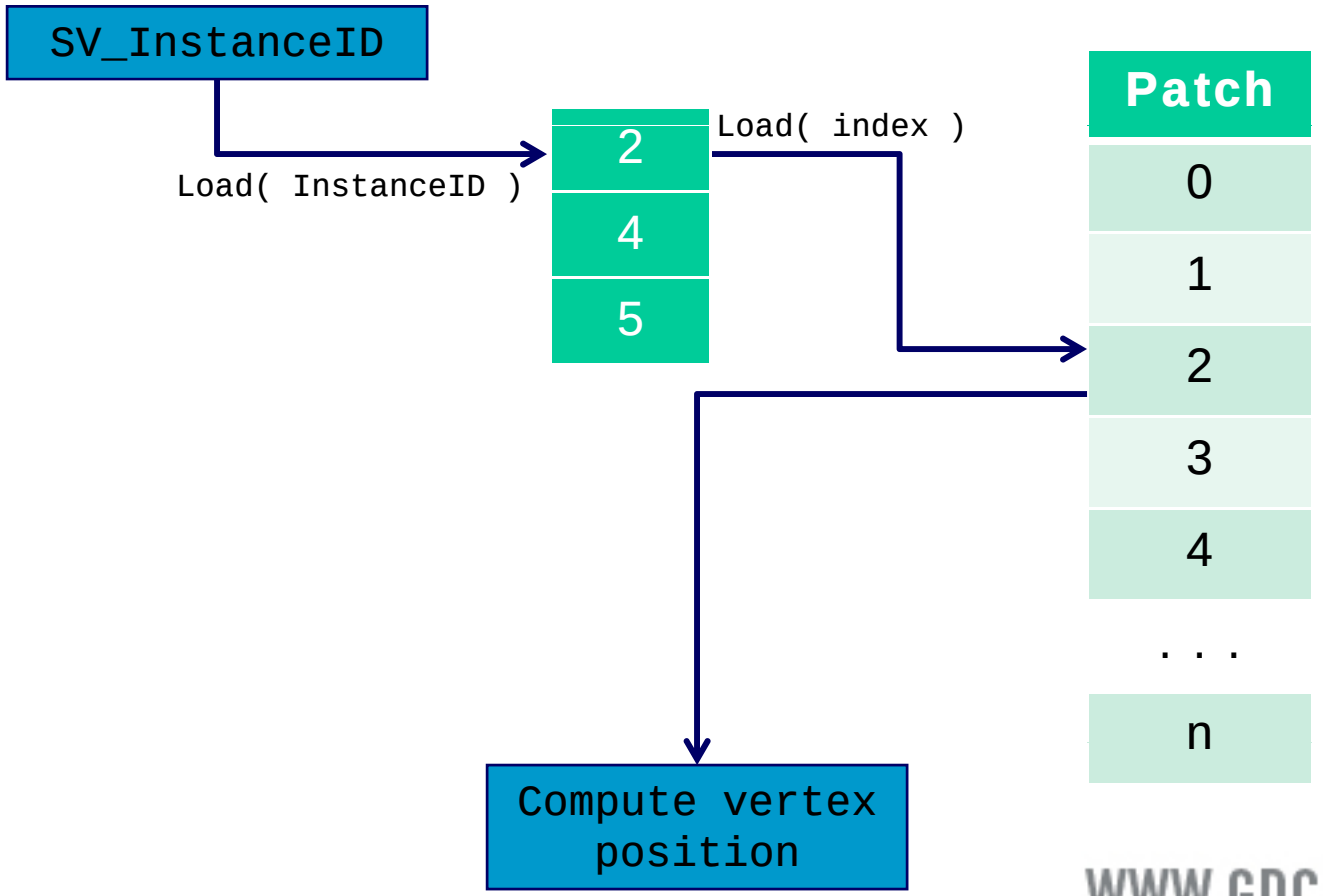
Per-patch factor

- Make several instanced drawcalls – one for each factor

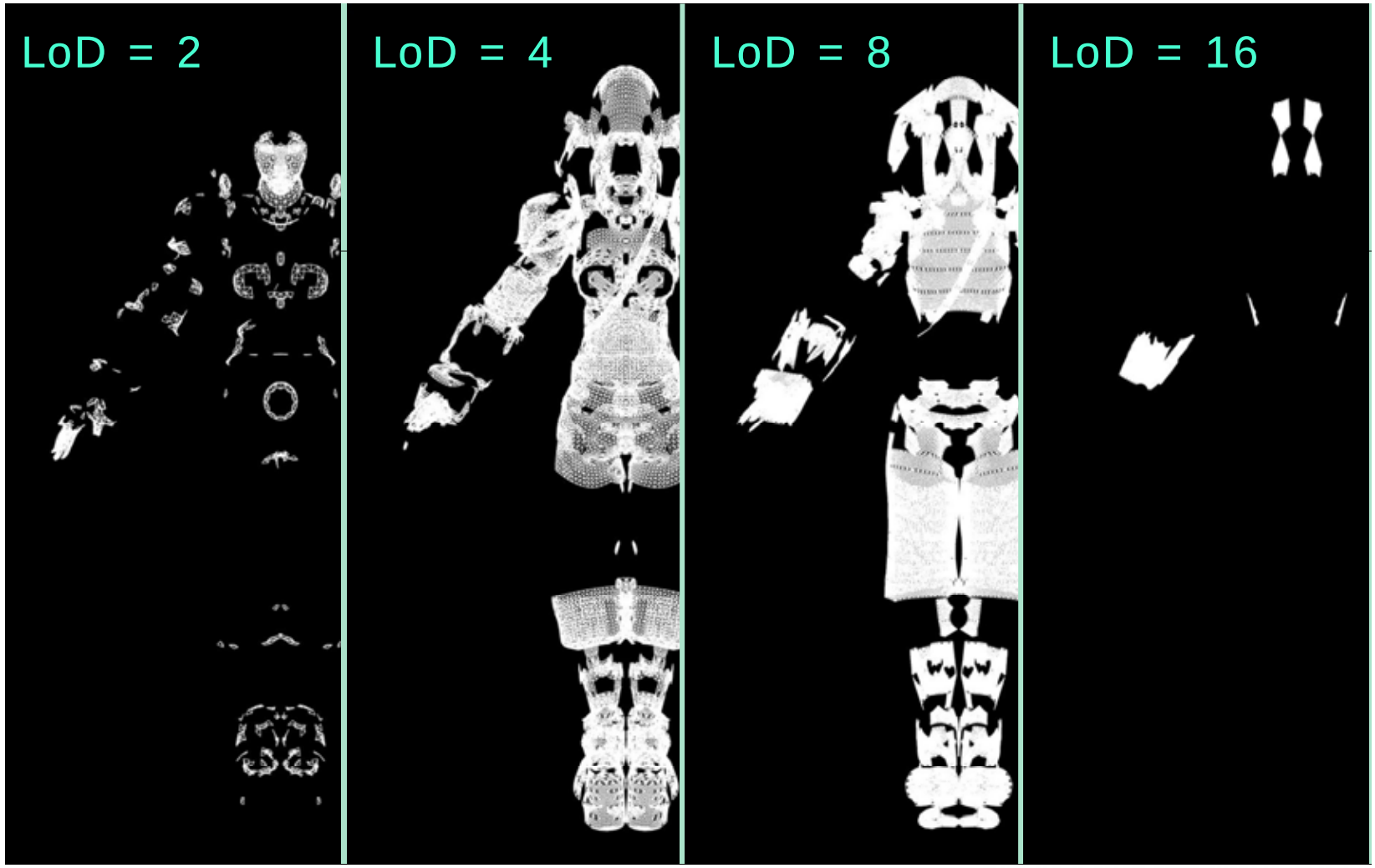


Pre-patch factor

- Use additional index buffer to fetch from patch array

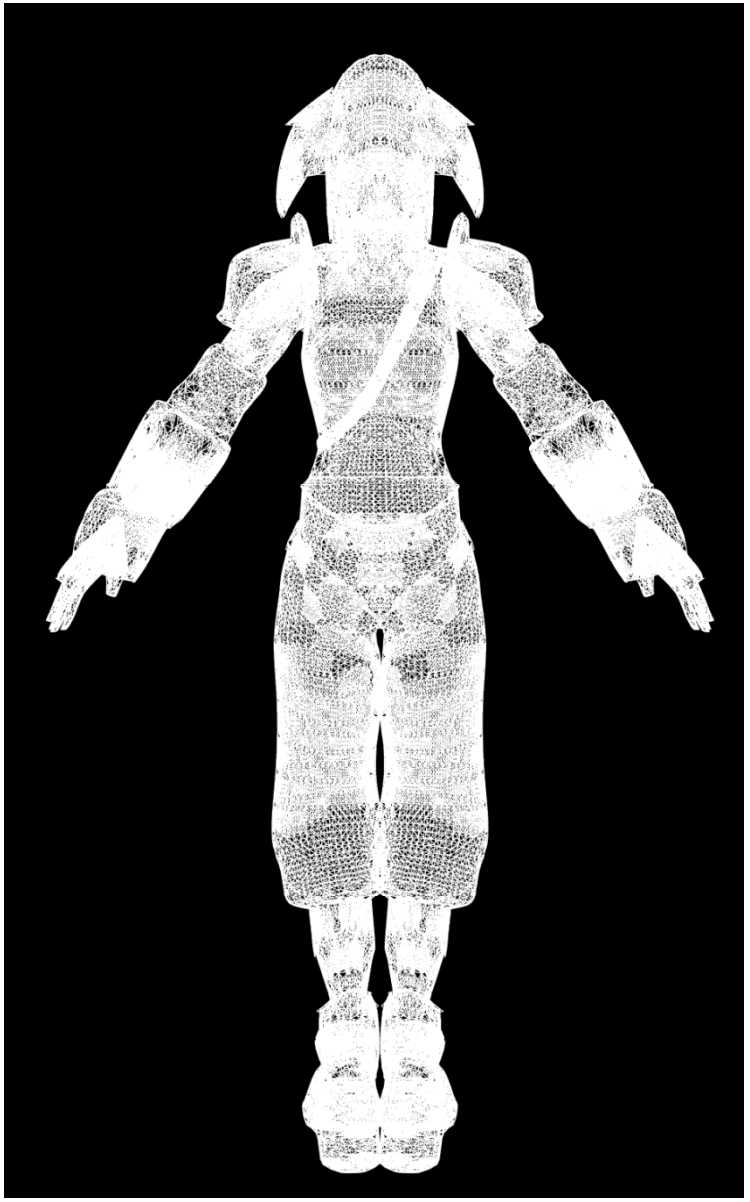


Per-patch factor



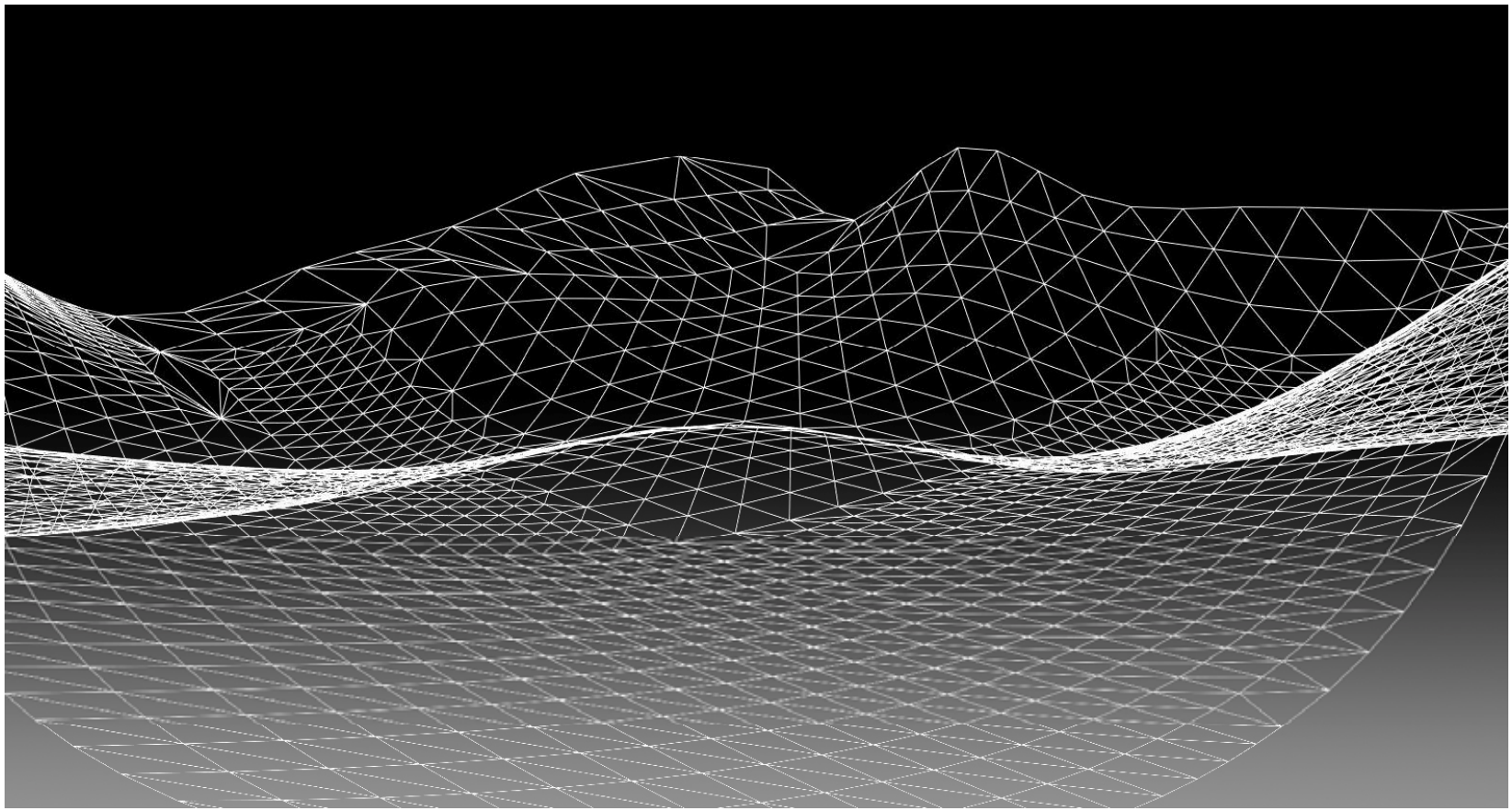
Per-patch factors

- After rendering patches for all factors, we get the final object rendered with LoD changing per-patch



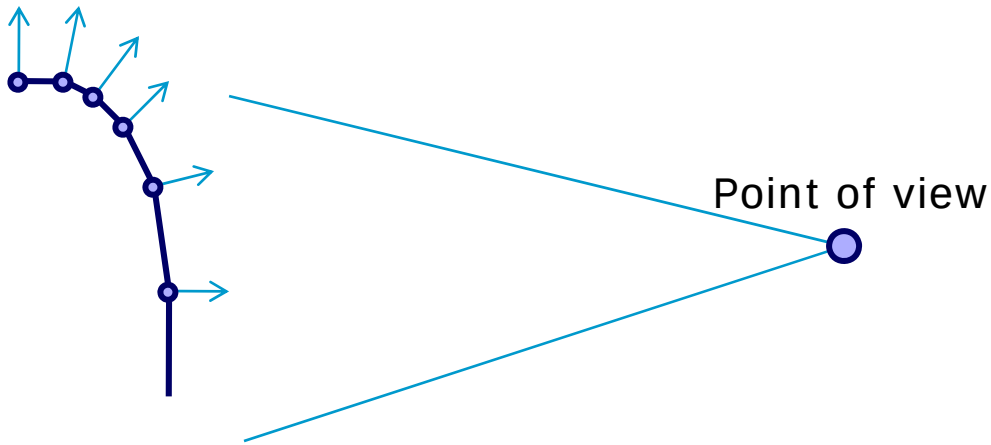
Per-patch factor

- Example of power-of-two per-patch tessellation factor:

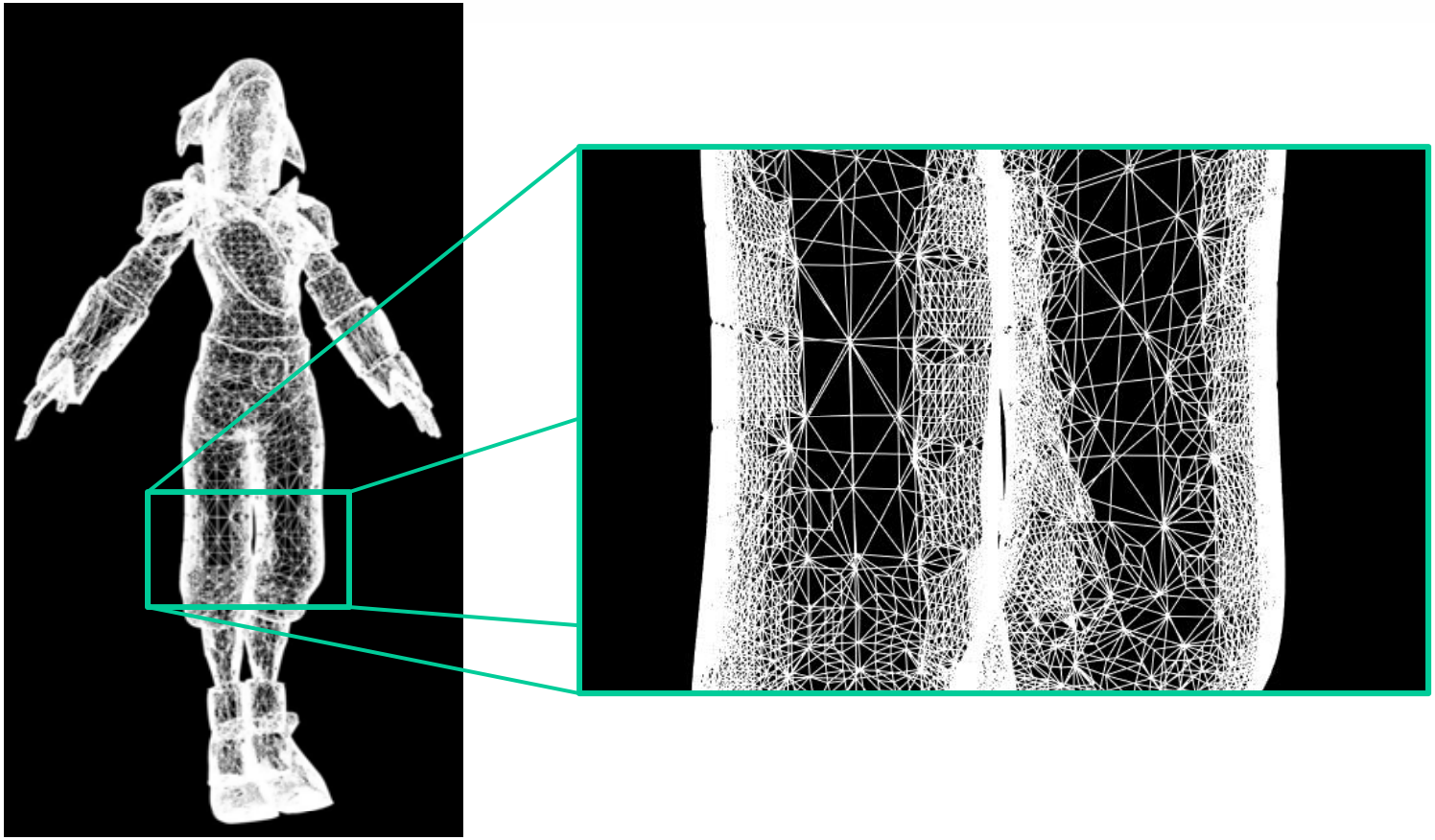


Silhouette-aware tessellation

- Increase tessellation factor when close to silhouette edges
- Refine the silhouette while the rest of the mesh remains coarse
- Good for rendering shadowmaps



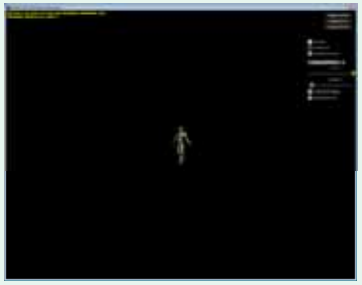
Silhouette-aware tessellation



Performance and conclusions

Performance comparison

- Mesh: **6118** patches, **256** vertices each, **8800 GT**

Distance	Dynamic tessellation with LoD	Pretessellated mesh
	39.32 FPS	36.70 FPS
	230.61 FPS	40.60 FPS

Memory consumption

Mesh: **6118** patches, **256** vertices each, **8800 GT**

	Coarse mesh	Height map	Diffuse map	**Total size**
Dynamically tessellated mesh	3 728 KBs	4 096 KBs	4 096 KBs	**11 920 KBs**

	Detailed mesh	Diffuse map	**Total size**
Pretessellated mesh	48 944 KBs	4 096 KBs	**53 040 KBs**

Conclusions

- Tessellation allows to achieve outstanding visual appearance while increasing rendering efficiency
- Instanced tessellation enables tessellation on today's hardware, giving access to unique tessellation features
- **Start thinking and experimenting today!**

Big thanks to:

- Ignacio Castaño
- Kirill Dmitriev

References

Generic Mesh Refinement on GPU

Tamy Boubekeur & Christophe Schlick

Efficient Tessellation on GPU through Instancing

Holger Gruen

Thanks!