D3D11 Software Tessellation

John Kloetzli, Jr
Graphics Programmer, Firaxis Games

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21
2014

About Firaxis

- Founded in 1996
- Strategy games!
- Sid Meier lead designer
- 20+ shipped games
  - Civilization V
  - XCOM: Enemy Unknown

"Games that stand the test of time"

About Me

- I work on the Civilization team
  - Graphics programmer
  - Over 7 years at Firaxis
  - Procedural modeling
  - Terrain rendering

SID MEIER'S CIVILIZATION V

Civilization V

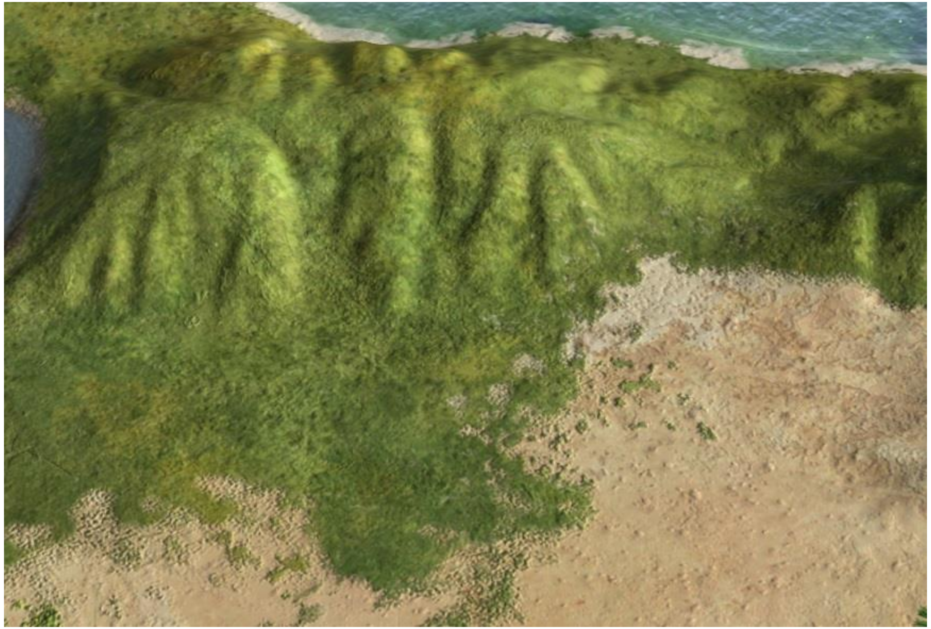
- Shipped Sept. 2010
- One of the first DX11 games
  - Variable-bitrate GPU texture decompression
  - Hardware tessellation
- Two large expansions
  - *Gods & Kings*
  - *Brave New World*

OLANO et al. **Variable Bit Rate GPU Texture Decompression**. In *EGSR 2011*

Civilization V

- Low-res Heightmap
  - 64x64 per hex
  - Procedurally generated
  - Unique – no repeat

- High-res Materials
  - 512x512 per hex
  - Artist-created
  - Repeats across the world



Better Terrain

- Problem: Sharp features
  - Low-res heightmap cannot display unique, high-res detail
- Solution: High-res heightmap
  - More data (Compression? Streaming?)
  - Efficient Tessellation

GPU Displacement Tessellation

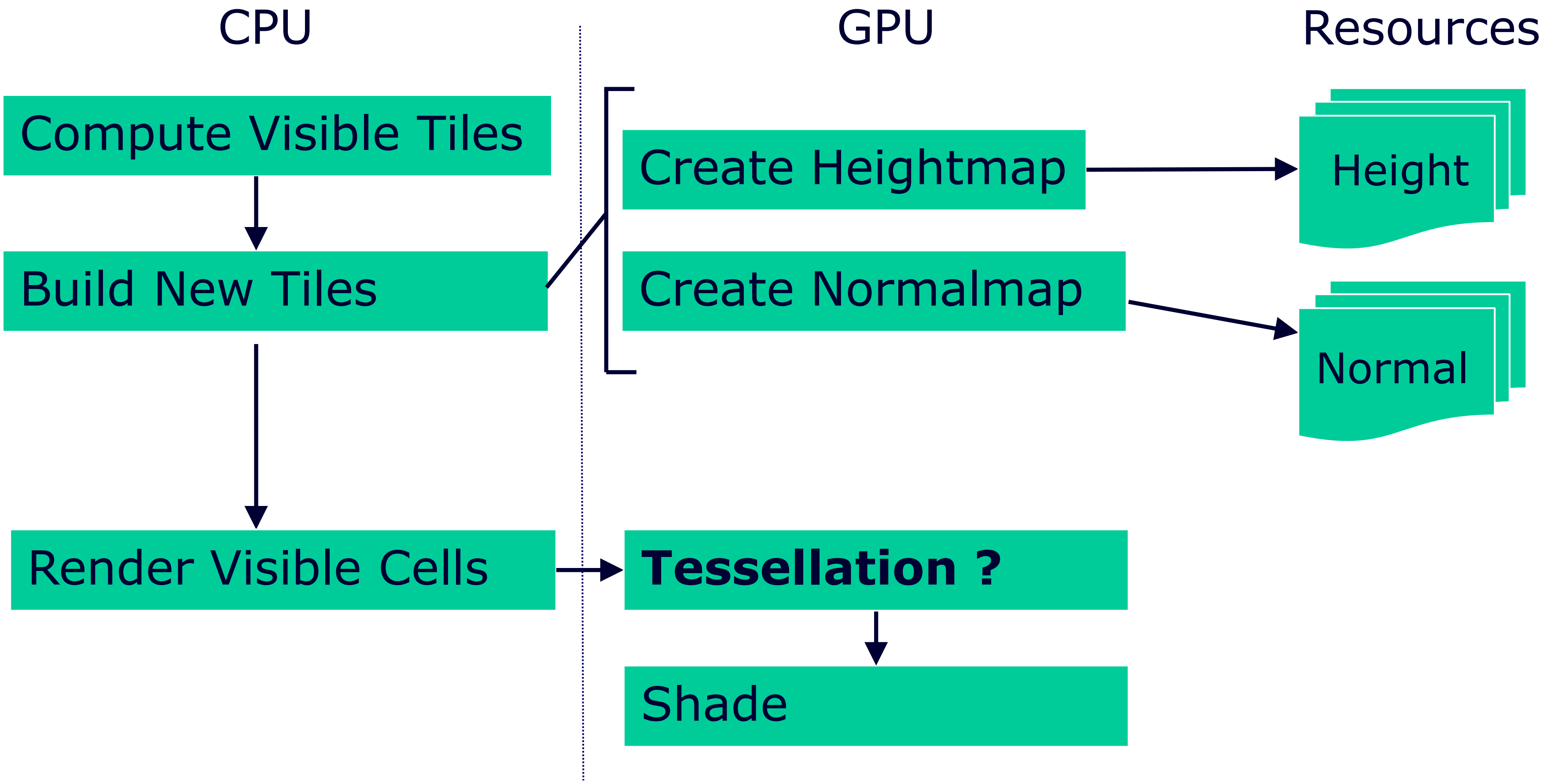
Demo

Simple procedural terrain...

- Ridges to test difficult case
- Assume strategy game camera (lots of pan/zoom)
- High res: *256x256 Heightmap per tile*
- Large: *128x128 tiles (32,768x32,768 heightmap)*

...all done on the GPU

- Heightmap/Normalmap created on demand
- Use texture arrays to implement megatexture
- **Tessellation created on demand using GPU**



CPU

GPU

Resources

Compute Visible Tiles

Build New Tiles

Render Visible Cells

Create Heightmap

Create Normalmap

Tessellation ?

Shade

Height

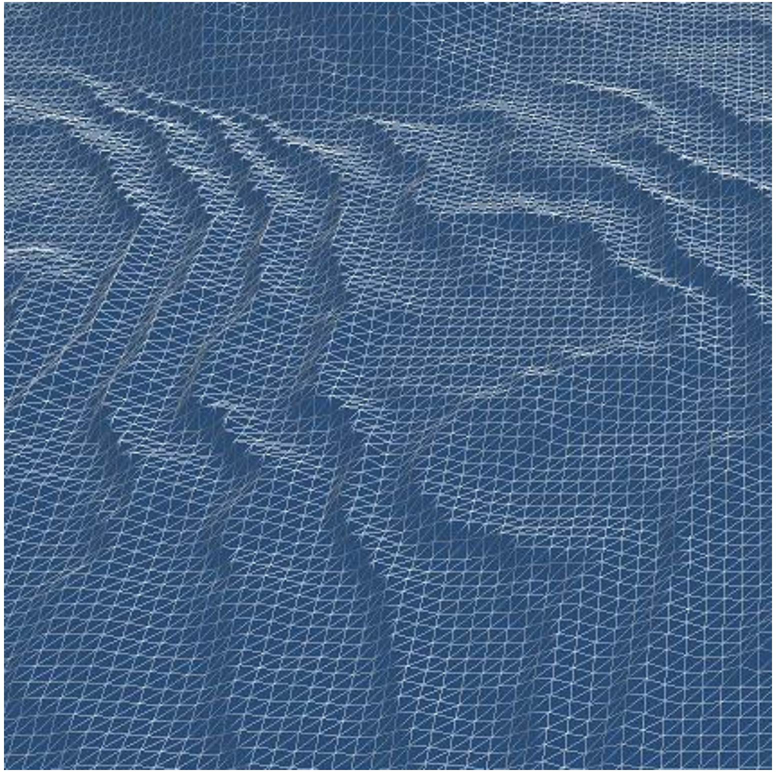
Normal

Overview

- Fixed Tessellation
  - Spoiler: Doesn’t work well
- Hardware Tessellation
  - Easy to implement
  - Better performance
  - Questionable quality
- Variable Software Tessellation
  - Complex to implement
  - Great quality/performance balance

Fixed Tessellation

- Pre-tessellate fixed-res mesh
  - Render same mesh for each cell
  - Displace in VS
- High-res is slow
  - Lots of geometry (IA/Memory)
  - Tiny triangles (Quad utilization)
- Low-res is ugly
  - Triangles do not match data

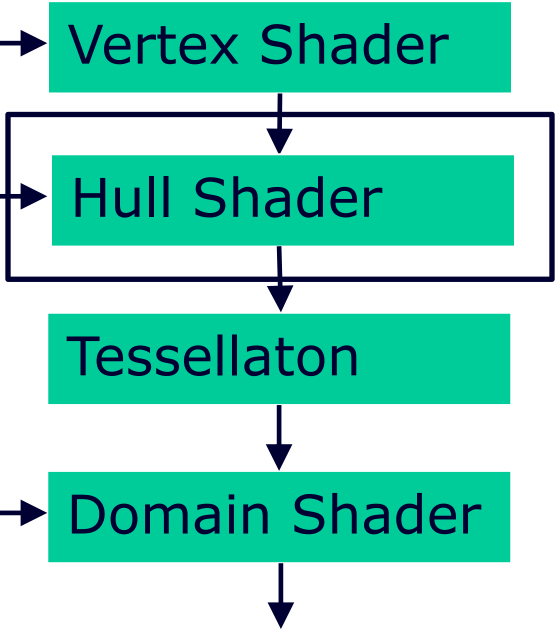


Hardware Tessellation

Low res patch mesh
(Lower IA/Memory load)

Subdivision levels
Level for patch
Level for each edge
(Match with neighbors)

Displacement here
(same vertex load)

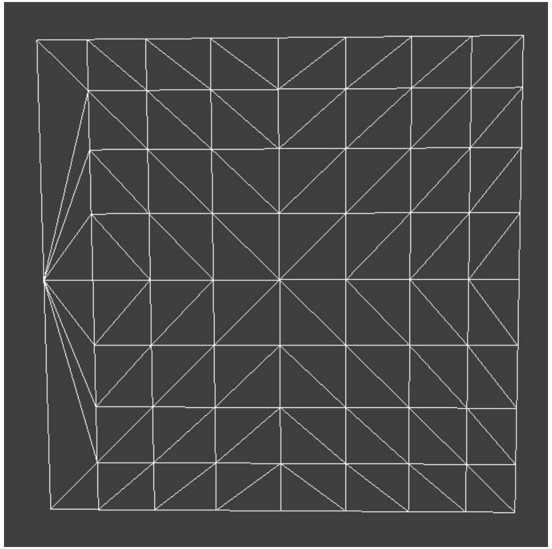


Vertex Shader

Hull Shader

Tessellaton

Domain Shader



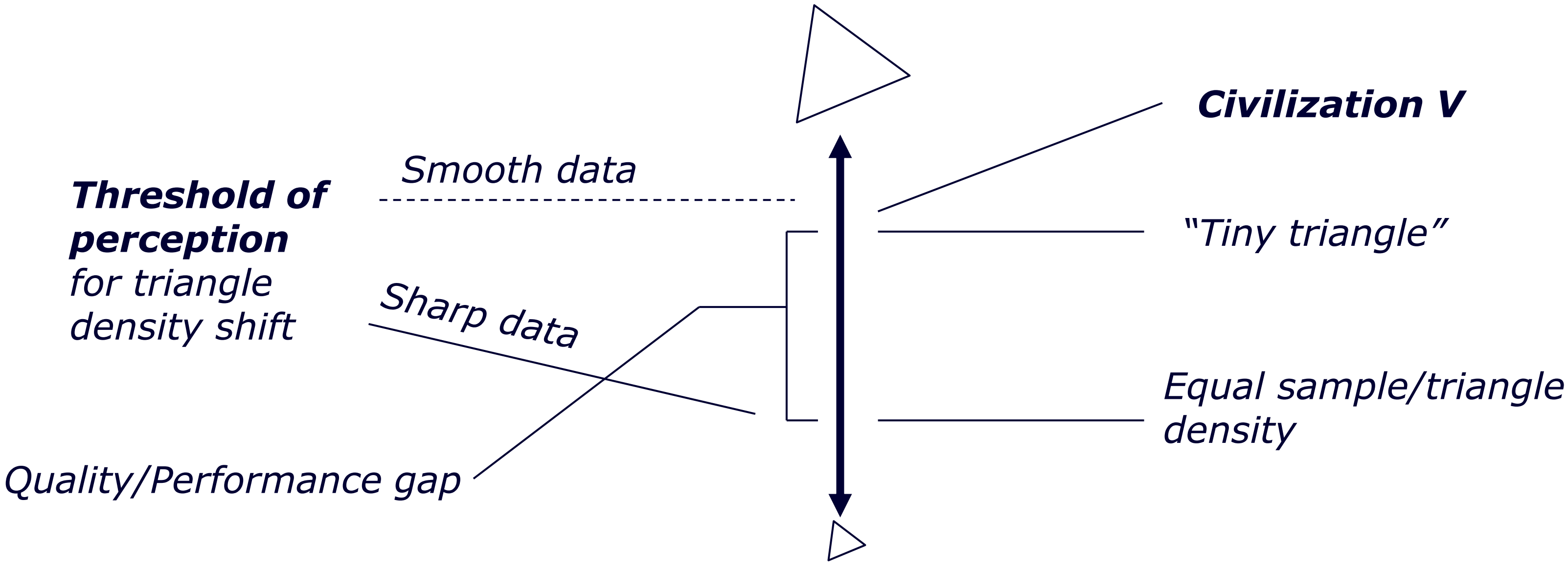
Hardware Tessellation

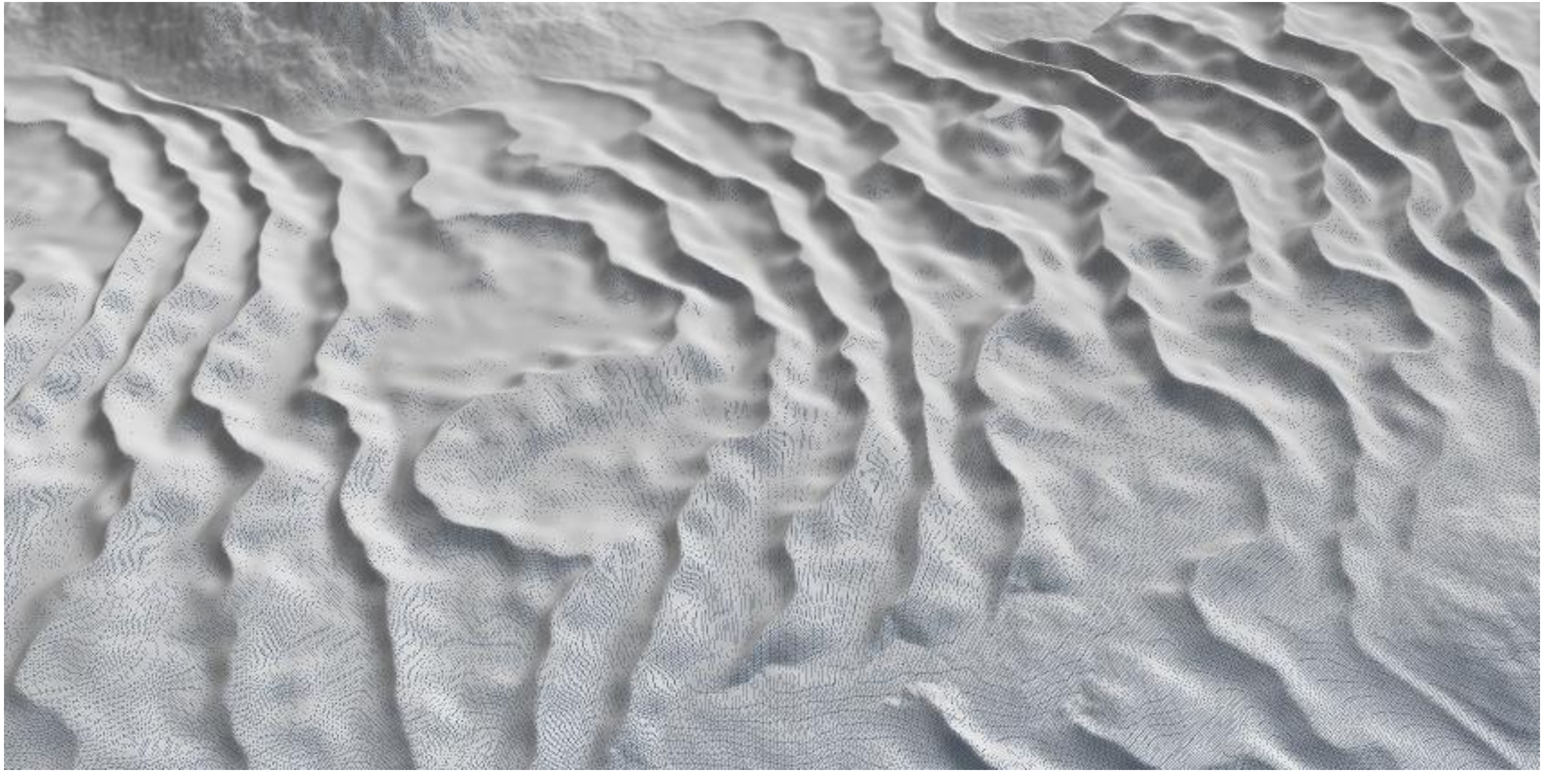
- Continuously variable tessellation levels
    - Complex resampling of displacement map
    - Blurring - high frequency data disappears
    - Aliasing - “Sliding” or “Shifting” artifacts
- Power-of-two tessellation levels
    - Much easier sampling of displacement map
    - Hard to change tessellation level without “popping”

View-Based Hull Shaders

- Use camera information to set tessellation level
  - Distance from camera
  - Height of camera (Civ V) *best for strategy games*
  - Projected screen size
  - Silhouette enhancement
  - …and variations

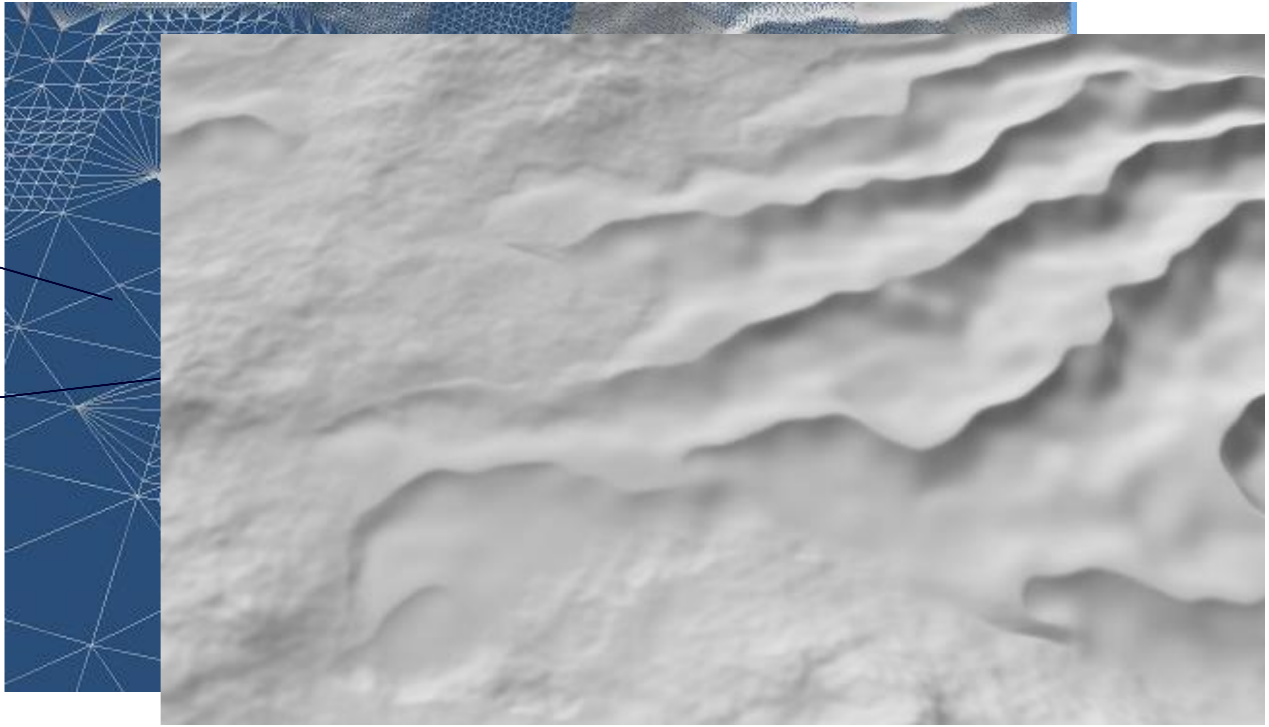
View-Based Hull Shaders


Civilization V
Smooth data
Threshold of perception for triangle density shift
"Tiny triangle"
Sharp data
Equal sample/triangle density
Quality/Performance gap



Quad covers 1x1 height samples

Data-Based Hull Shaders

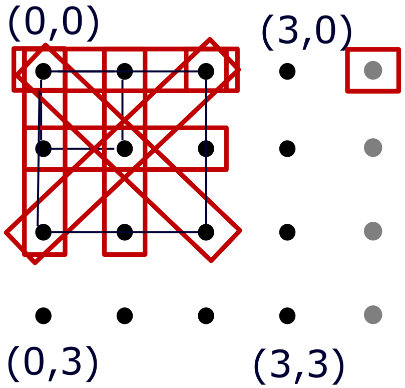


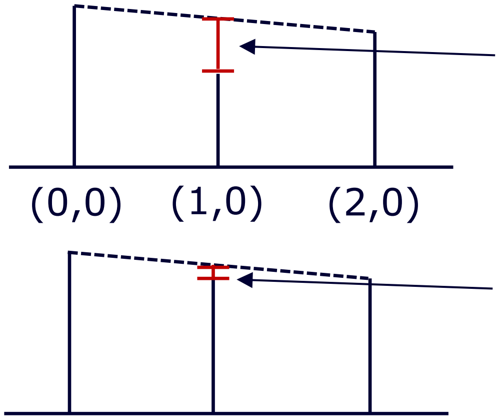
Smooth Patch

Sharp Patch

Data-Based Hull Shaders

- Does quad (0,0)x(1,1) contribute to the final image?
  - *We can easily run this test at power-of-two resolutions*
    - At level N skip 2^N samples
    - Increase threshold at each resolution (Demo: Multiply by 1.7)





Large delta is **over** threshold, does contribute

Small delta is **under** threshold, does not contribute

Data-Based Hull Shaders

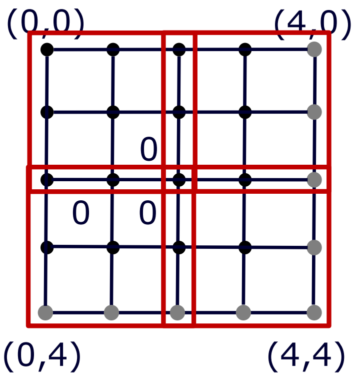
- Build MIP hierarchy of ‘necessary’ quads
  - Run compute kernel across each level
  - Results in tessellation level for patch

    Since we limited ourselves to pow2 tessellation

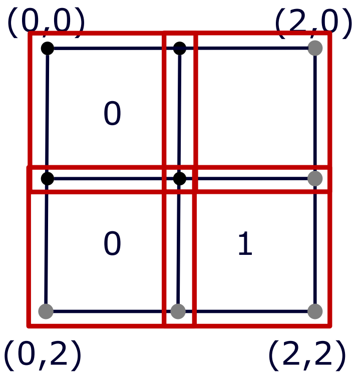
Kernel:

```
if lower level quad marked,
  output lower level
else if this quad passes test
  output this level
else
  output nothing
```

Level 0

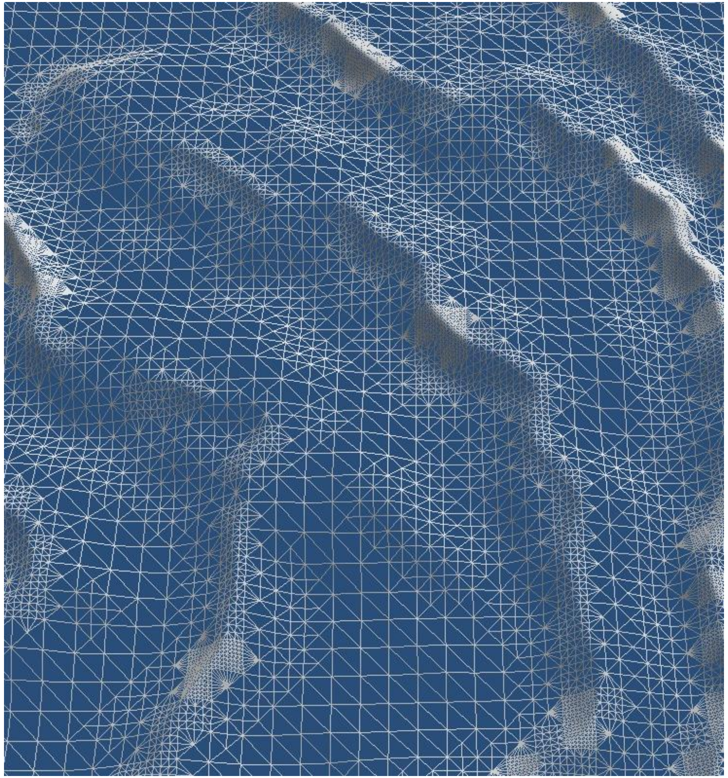


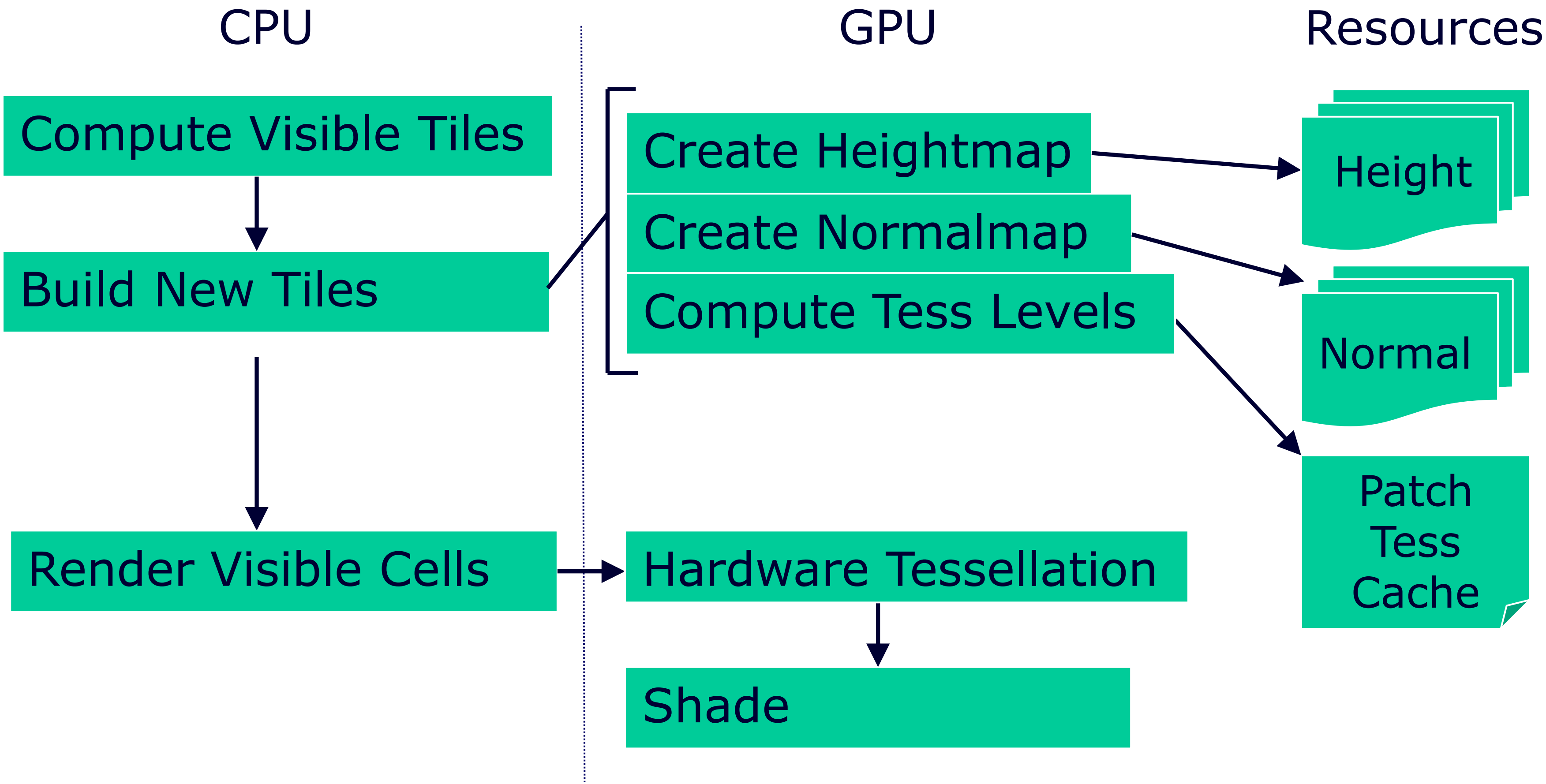
Level 1



Data-Based Hull Shaders

- In demo…
  - Higher resolution
    - Cell size is 256x256
    - 16x16 patches per cell (fastest)
  - Cache tessellation levels
    - Compute when tile becomes visible
    - Large cache texture stores all tessellation levels
  - Use Compute Shaders…
    - To generate the level heirarchy
    - To copy highest level into cache texture
  - Use Hull Shader…
    - To lookup tessellation level for patch
    - To match tessellation with neighbors





CPU

GPU

Resources

Compute Visible Tiles

Build New Tiles

Render Visible Cells

Create Heightmap

Create Normalmap

Compute Tess Levels

Hardware Tessellation

Shade

Height

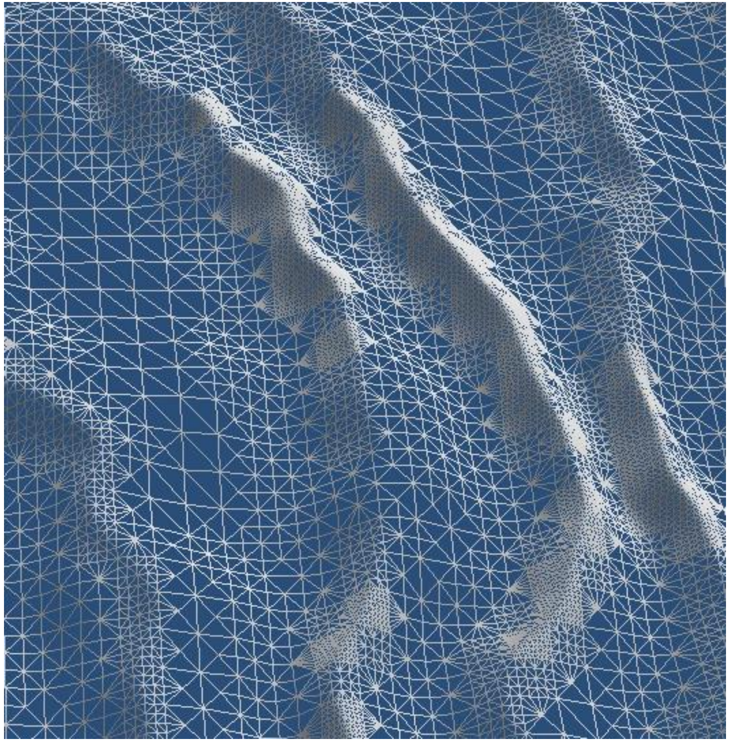
Normal

Patch Tess Cache

Data-Based Hull Shaders

- Pros
  - Looking at the heightmap was key
  - Many fewer tiny triangles generated
  - High quality (no compromise)
- Cons
  - Need to compute+store tess levels
  - Does not match data closely
    - Patch positions are fixed
    - Patch dicing pattern fixed
    - Still many tiny triangles

Can we find a better solution for our use case?

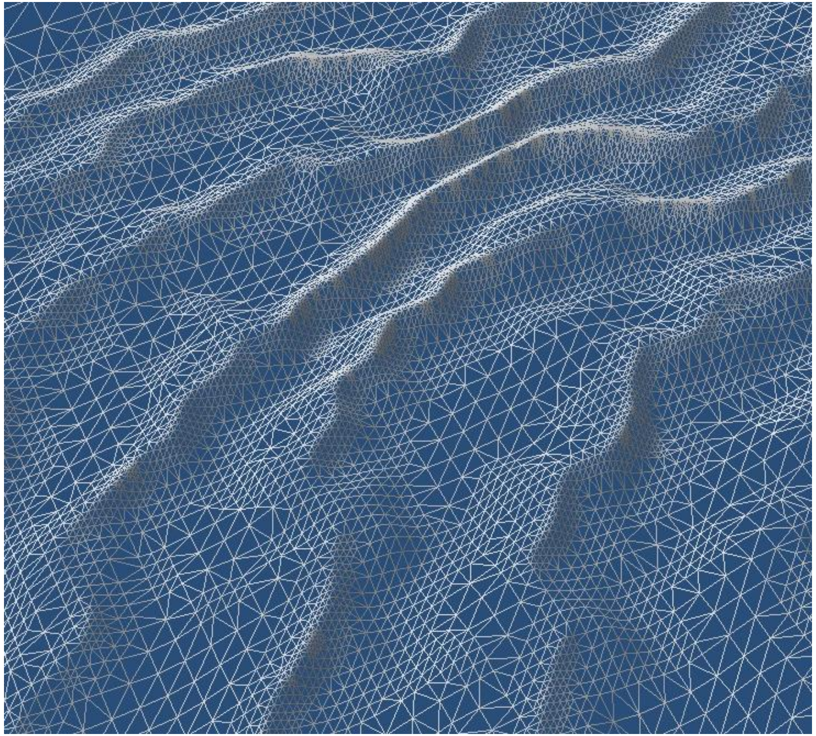


Software Tessellation

- Inspiration: *AdaptiveTessellationCS40*
  - D3D11 DirectCompute sample from Microsoft
  - Simulate hardware tessellation in software
  - Run in D3D11 Downlevel 10.0
  - Goal: Increase the reach of D3D11-style tessellation
- Why not design a new tessellation algorithm?
  - Custom-built for detailed terrain rendering
  - Custom-build for strategy games
  - Run in compute shaders

Software Tessellation

- Design goals:
  - Avoid tiny triangles
  - High quality
  - *Efficiency (for real-time)*
- Our solution:
  - Simplify patch definition
  - Generate more patches
  - Data-based patch generation
  - Data-based patch dicing

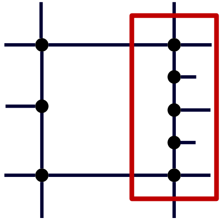


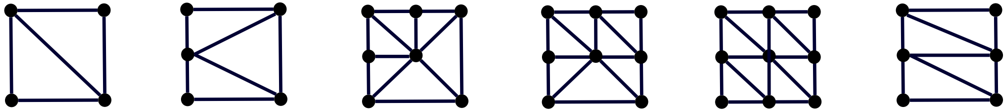
Software Tessellation

- Simplify patch definition
  - Only support pow2 patches
  - No tessellation factors for center
  - Edge tessellation factors 0 or 1
  - Patch defined by uint4

    [Position, Level, Dicing pattern]

Adjacent patches must be within one tessellation level





Only 16 possible patterns!

Software Tessellation

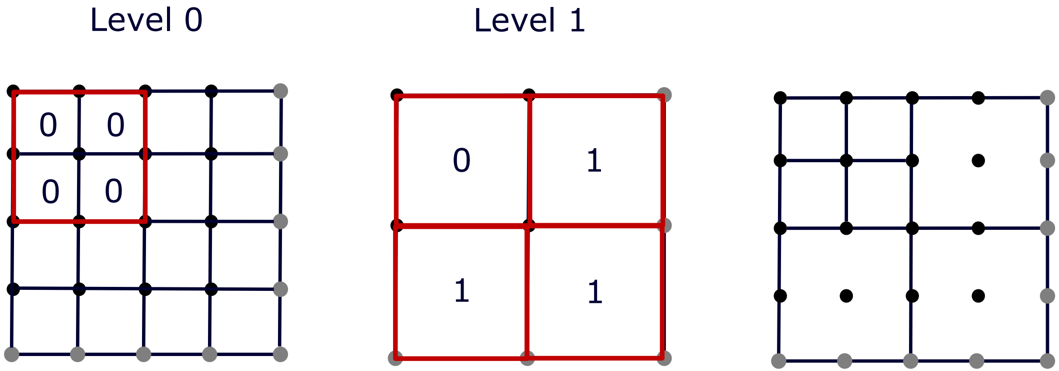
Kernel 1:

```
if lower level quad marked,
  output lower level
else if lower level neighbor marked
  output this level
else if this quad passes test
  output this level
else
  output nothing
```

Kernel 2:

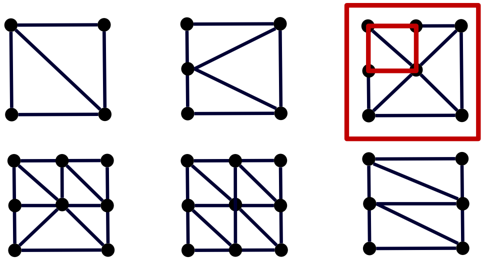
```
if any quad in group marked
  mark all quads in group
```

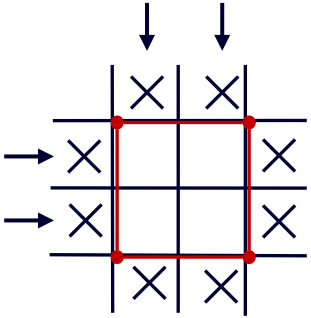
- Build Tess MIP hierarchy
  - Entire tile covered by patches
  - No overlapping patches
  - Adjacent patches within one level



Software Tessellation

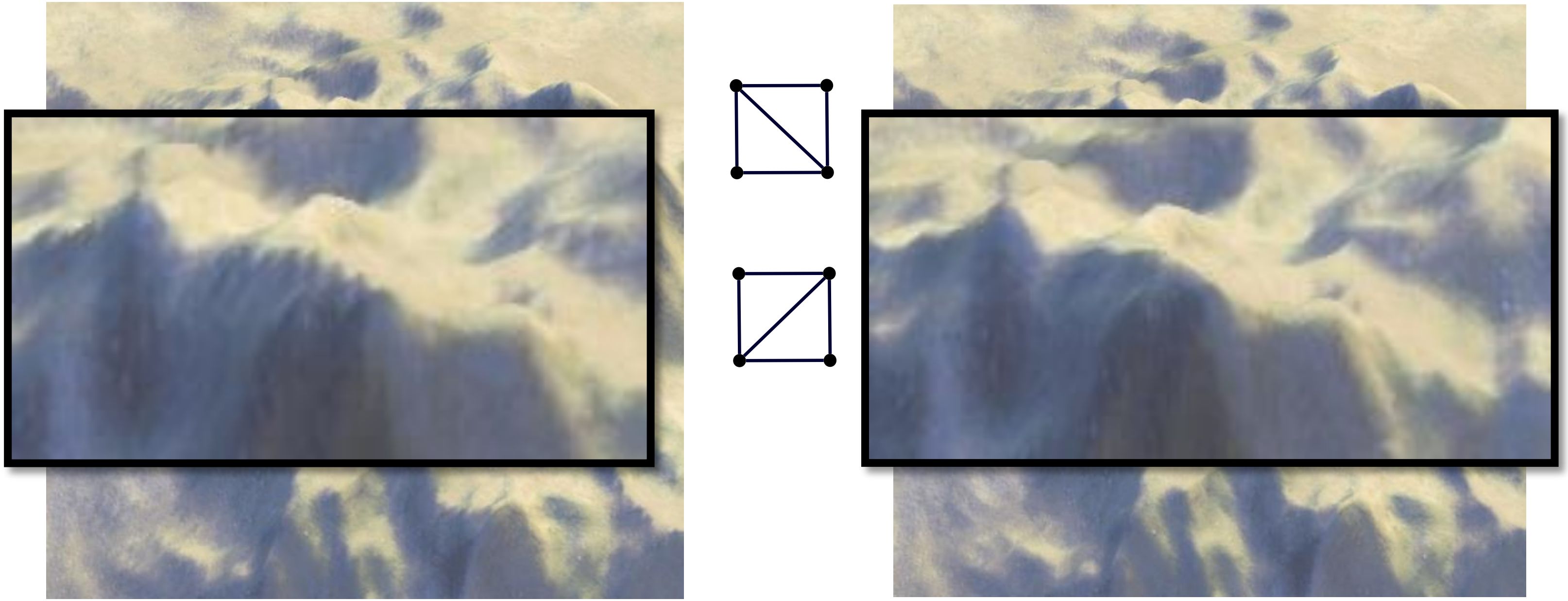
- Output patches by looking at MIP structure
    - Position, level from location with MIP
    - Look at lower-level neighbors to determine dicing pattern
    - Append to patch list
    - Optimization: *Break complex patches into component parts*





Software Tessellation

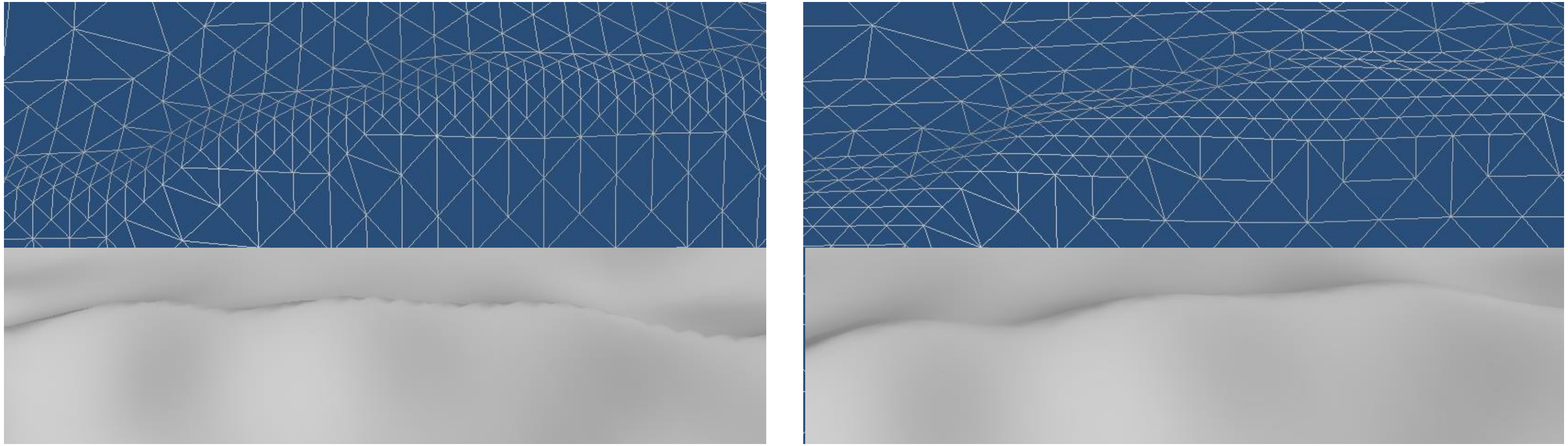
The direction we split quads is important



Extensions

- In our demo…
  - Treat patch split direction as separate dicing pattern
  - Process patch list to determine best split direction

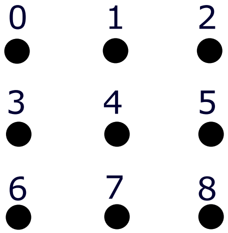
    Difference of normal (dot product)

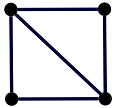


Software Tessellation

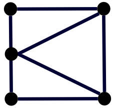
- How do we build geometry from patch list?

  Difficulty: Dicing patterns vary from 2 to 4 tris
- Simple algorithm: Degenerate geometry
  - Output 9 verts and 12 indices per patch
  - Extra verts and degenerate triangles not optimal
  - We are only getting indexing within a patch
  - Fast enough to run every frame





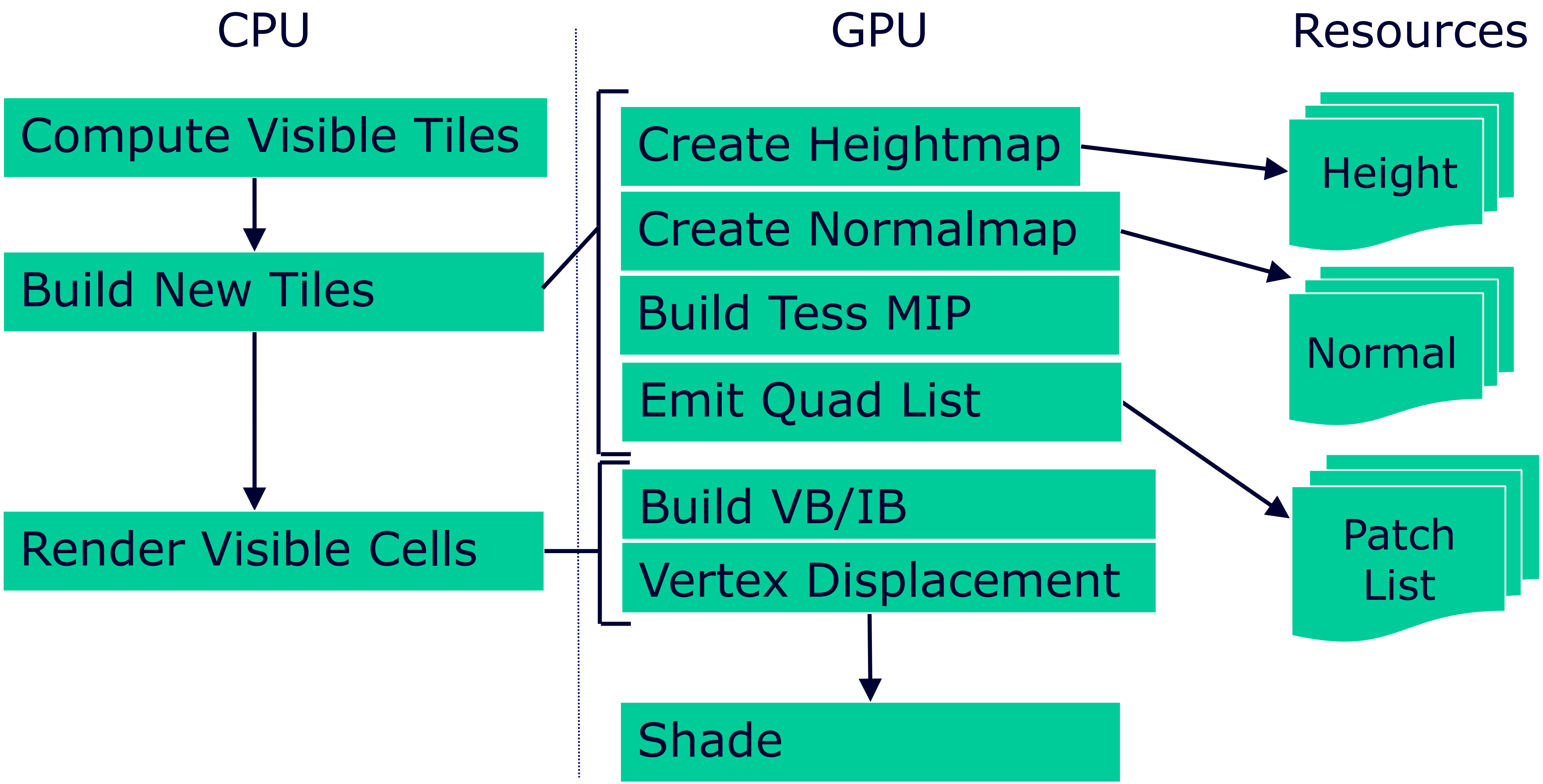
0	2	8	0	8	6	0	0	0	0	0	0



0	2	3	3	2	8	3	8	6	0	0	0

Software Tessellation

- How do we build **better** geometry from patch list?
- *AdaptiveTessellationCS40*
  - Use prefix-sum to get base vertex/index ID for each patch
  - Tightly packed VB/IB
  - Slower, indexing within patch only
- Tile Vertex ID table
  - Build table of all possible verts for an entire tile
  - Build verts that are referenced by any patch
  - Resolve vertex ID from table
  - Slowest, indexing across whole tile



CPU

GPU

Resources

Compute Visible Tiles

Build New Tiles

Render Visible Cells

Create Heightmap

Create Normalmap

Build Tess MIP

Emit Quad List

Build VB/IB

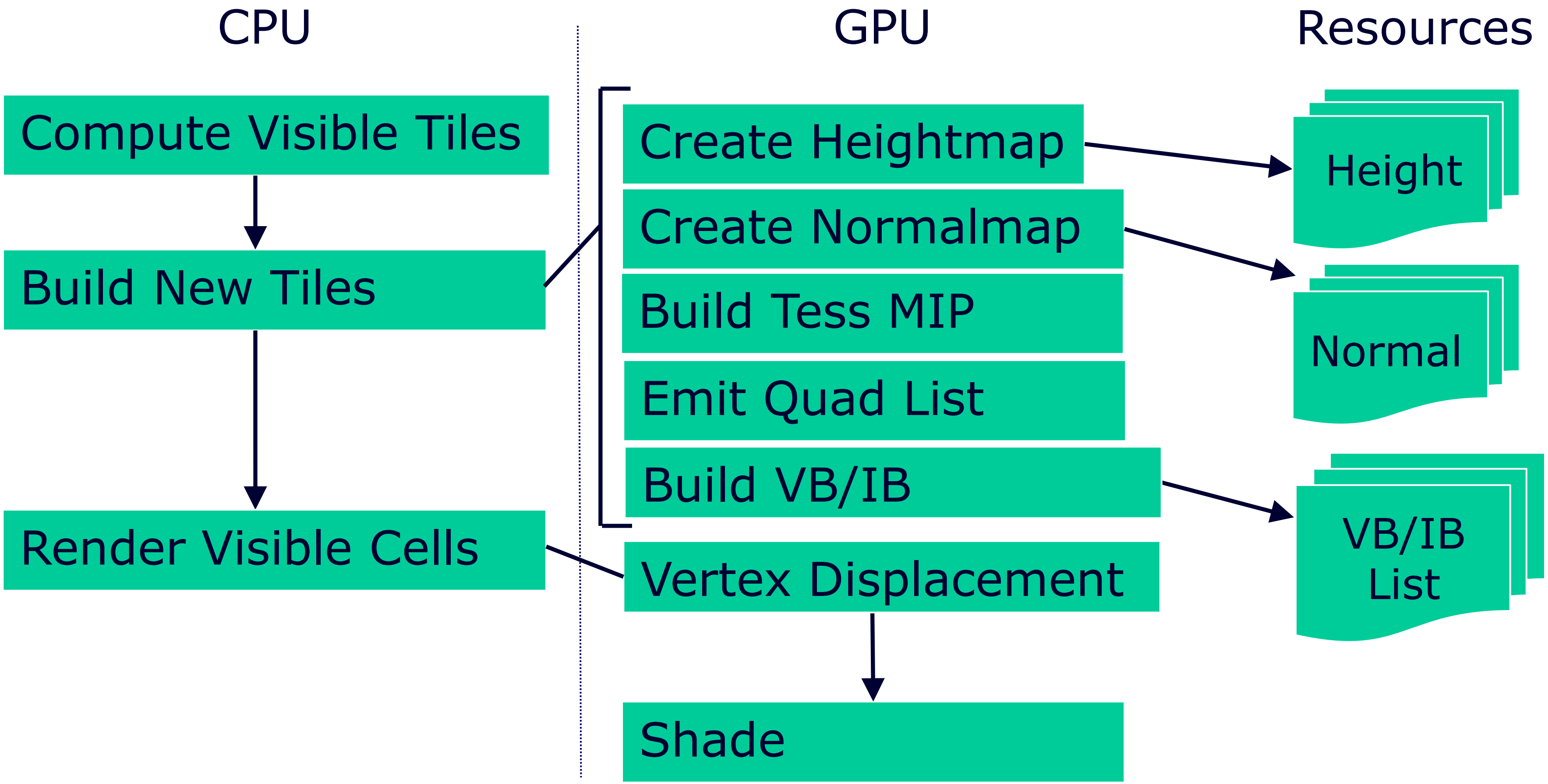
Vertex Displacement

Shade

Height

Normal

Patch List



CPU

Compute Visible Tiles

Build New Tiles

Render Visible Cells

GPU

Create Heightmap

Create Normalmap

Build Tess MIP

Emit Quad List

Build VB/IB

Vertex Displacement

Shade

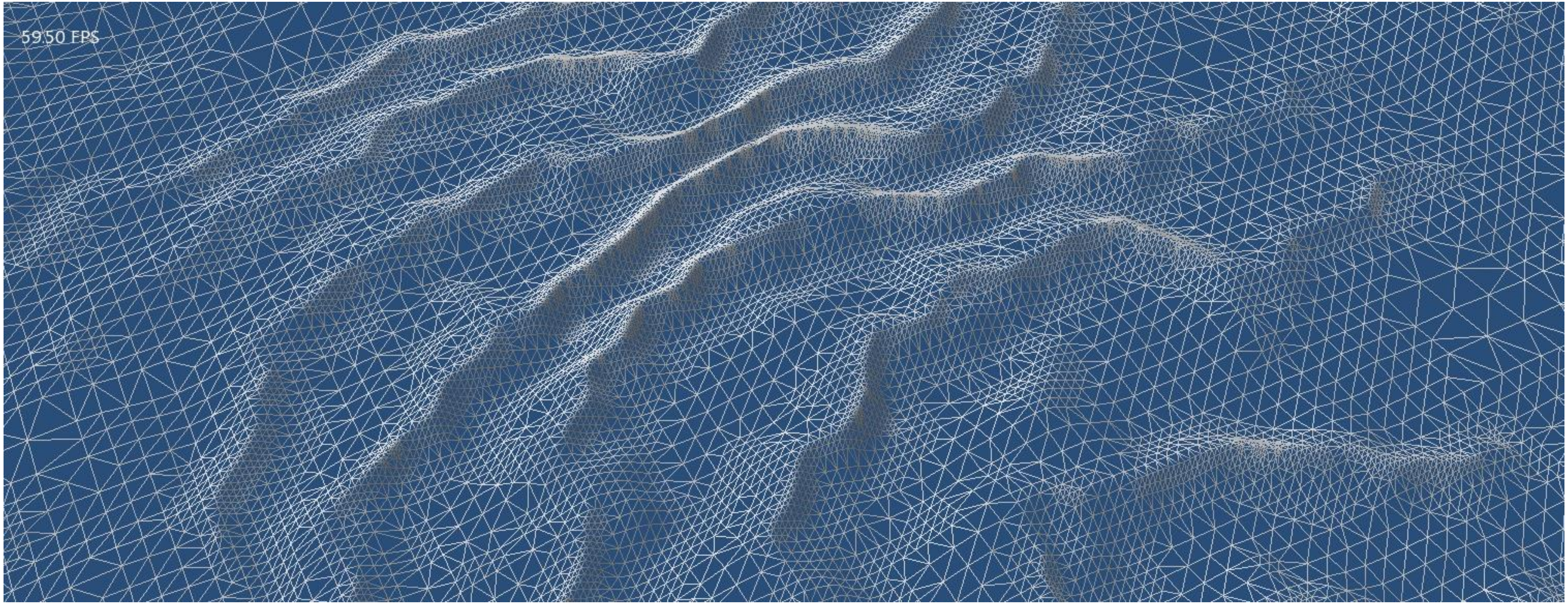
Resources

Height

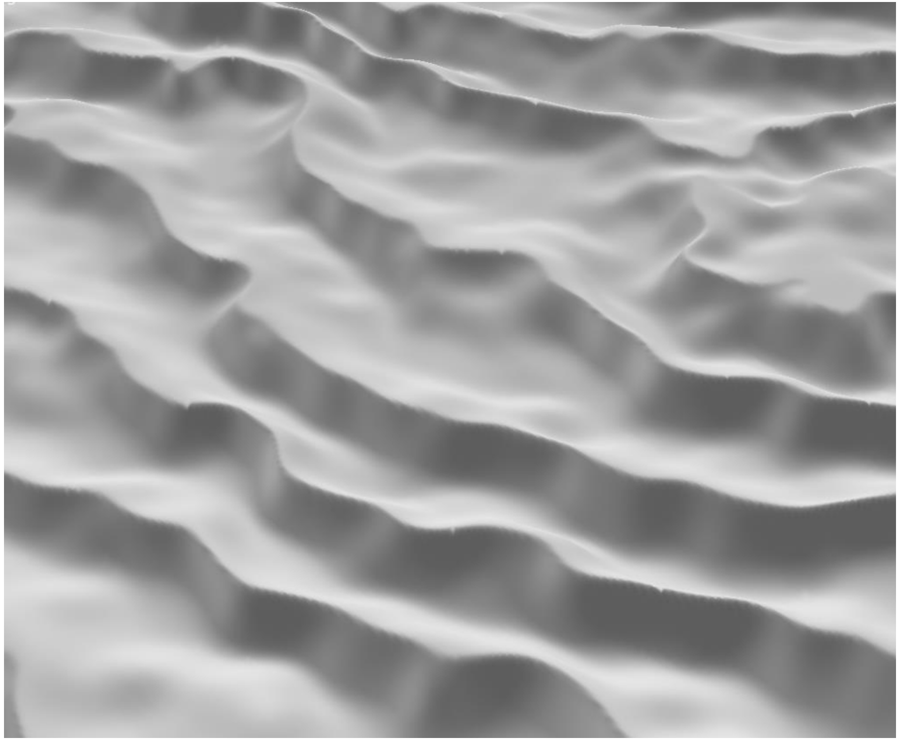
Normal

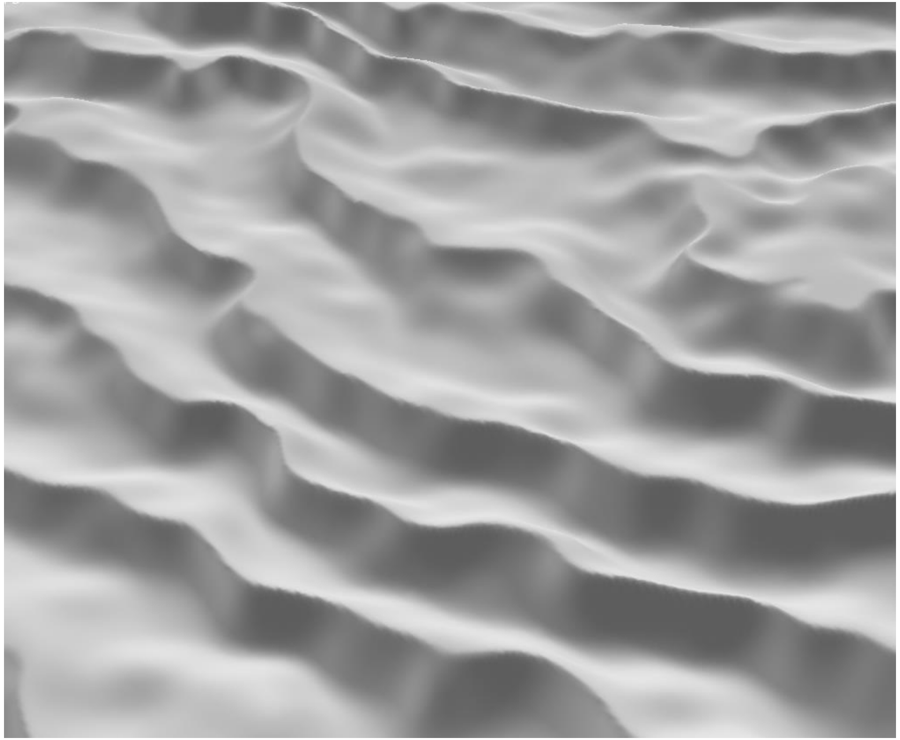
VB/IB List

Software Tessellation



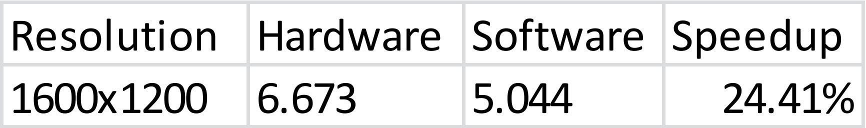
Software Tessellation





Software Tessellation

- Performance Results
  - AMD A10 APU/8670D GPU
  - Final render performance
  - GPU processing time for frame, ms
- Pros: Good performance, high quality
- Cons:
  - MIP heirarchy more complex + larger
  - Need patch list for every visible tile



Resolution	Hardware	Software	Speedup
1600x1200	6.673	5.044	24.41%

GPU PerfStudio2

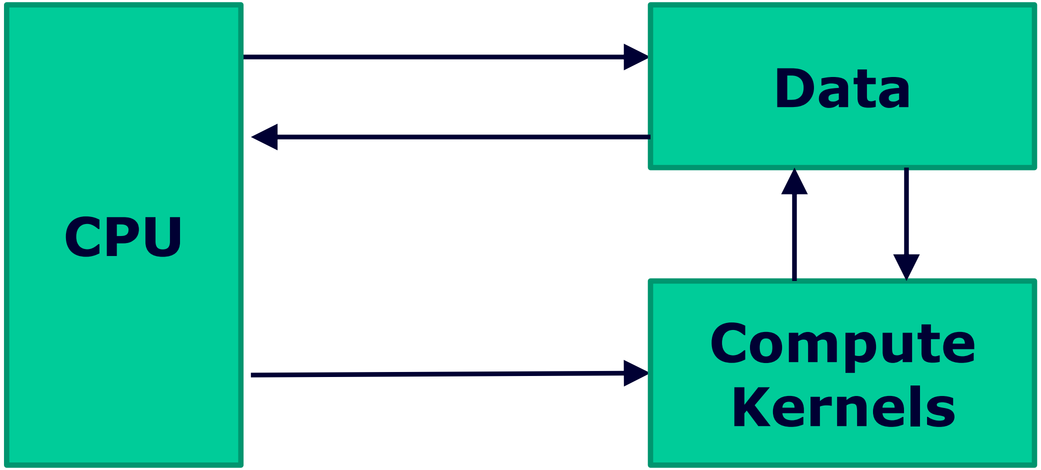
Conclusion: *Pixel shader execution dominates runtime, so it is worth doing extra work at the geometry level to generate efficient triangles.*

Implementation Tips

- Compute shaders have pros and cons
  - Generally very fast, but can be slower than PS (*texture swizzle patterns*)
  - Can run asynchronously on some hardware
- Atomic Operations vs. Atomic Counters
  - Atomic operations are general but slow
  - Atomic counters only increment or decrement…
  - …but have hardware backing on some systems
- Indirect draw/dispatch
  - Function parameters pulled from GPU buffer
  - Works well for draw calls (*Parameter is number of verts*)
  - Harder to use for dispatch (*Parameters are number of threadgroups*)

Conclusion

DX11: It’s all about compute!

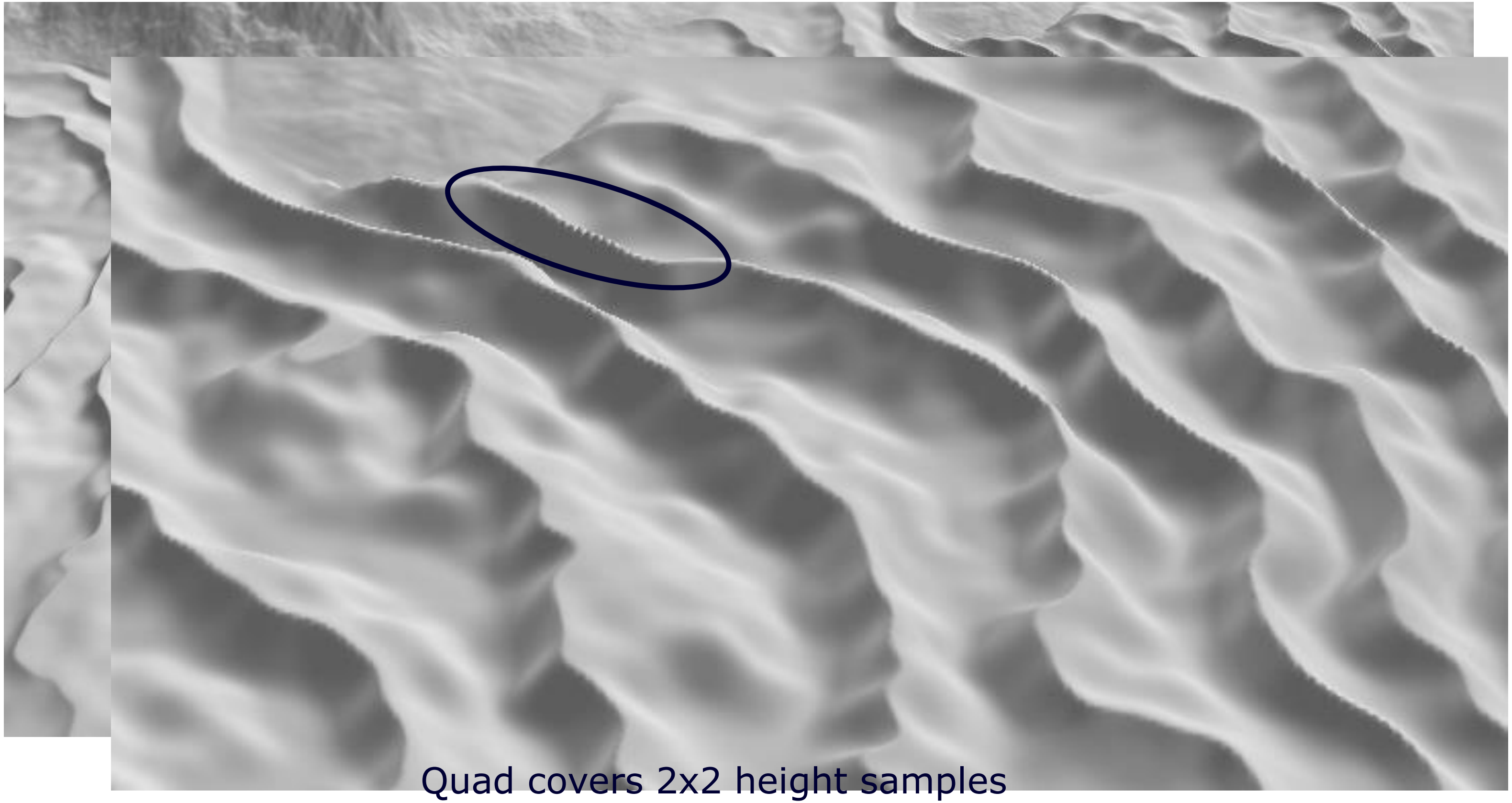


Questions?

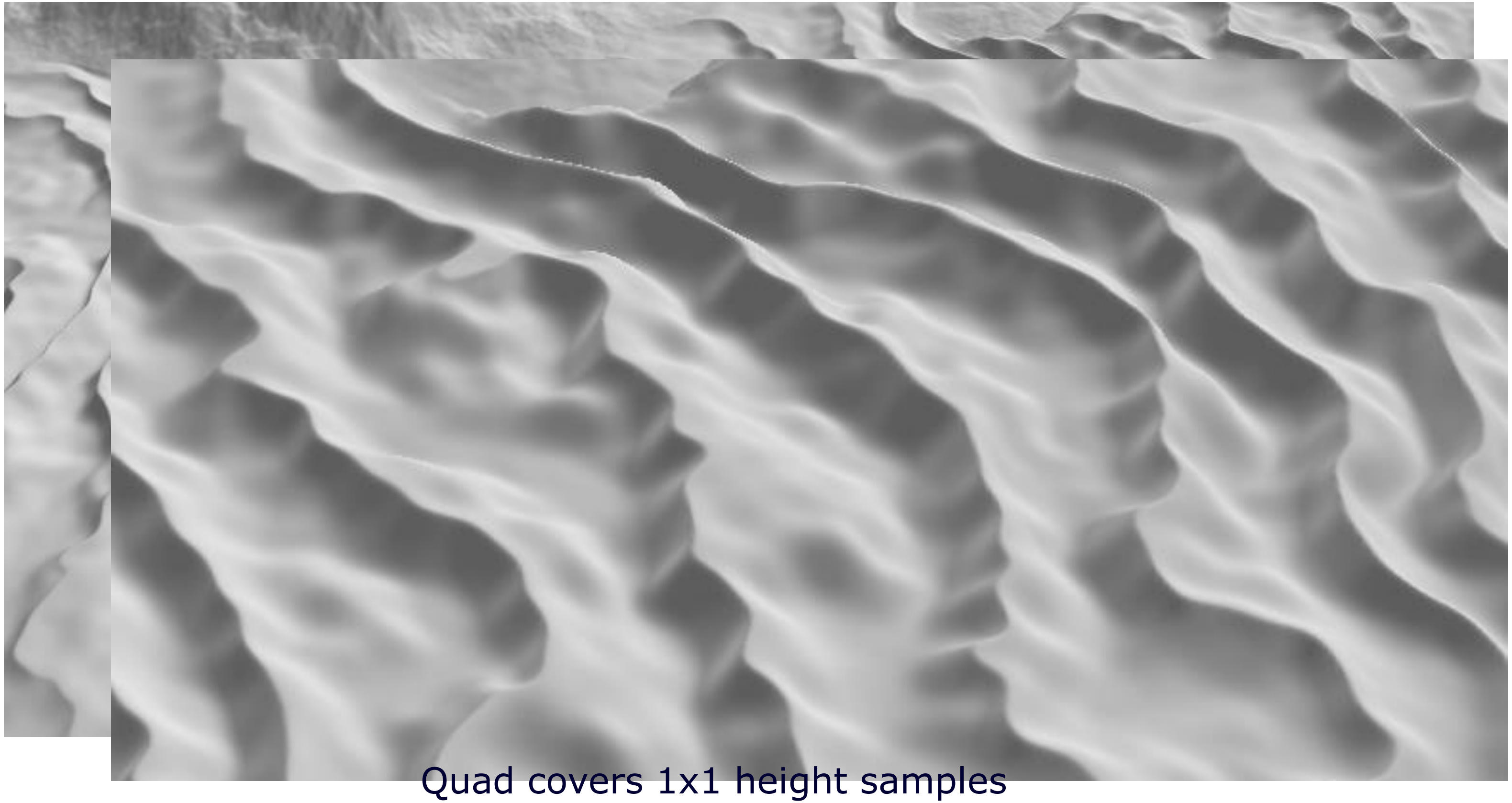
FIRAXIS GAMES

We are hiring!
www.firaxis.com

2K



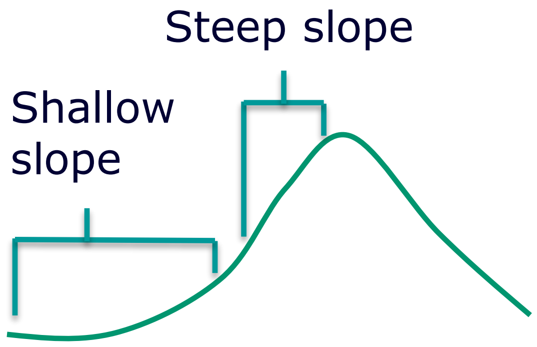
Quad covers 2x2 height samples

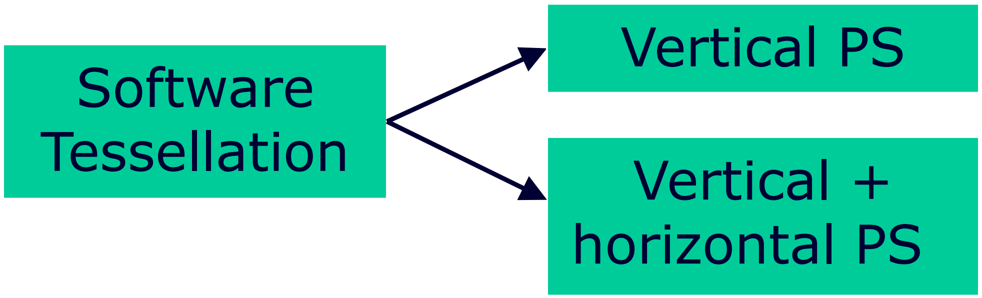


Quad covers 1x1 height samples

Extensions

- Take advantage of flexible geometry generation
  - Create more than one VB based on pixel shader needed
  - Can be huge optimization!




Software Tessellation

Vertical PS

Vertical + horizontal PS