Interactive Methods in Scientific Visualization

Terrain Rendering

Christian Dick

dick@in.tum.de

Computer Graphics & Visualization Group, Technische Universität München, Germany

IEEE Pacific Visualization Symposium 2009
Beijing, China

Outline

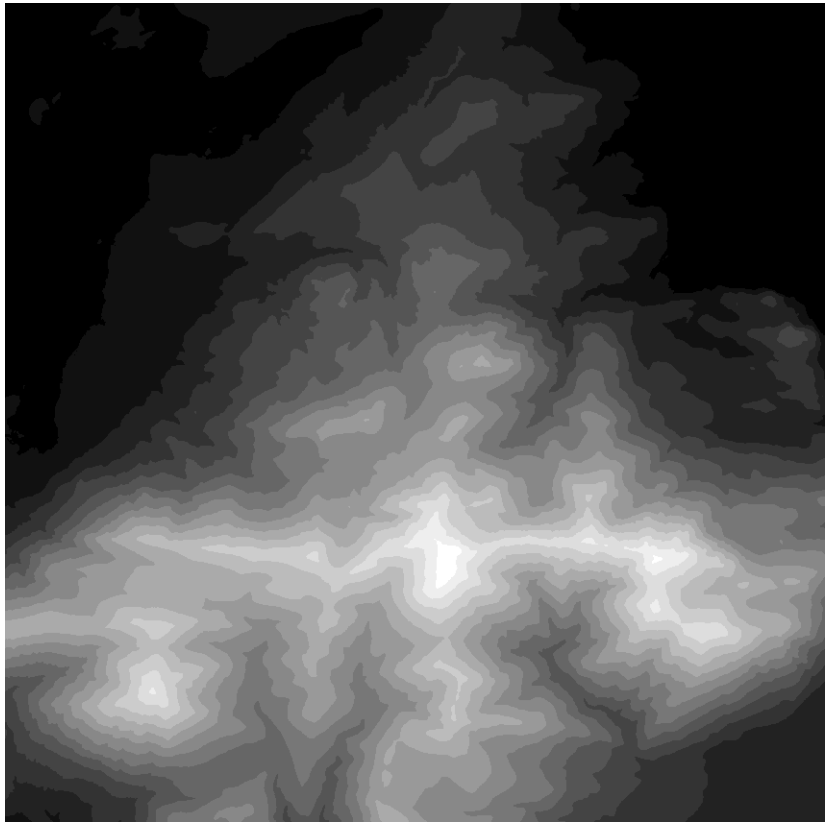
- Terrain Data and typical Storage Requirements
- Application Areas and their Requirements
- Overview of Continuous Level of Detail Terrain Rendering Techniques
- Texture and Geometry Compression
- Overlaying 2D Vector Data
- Demo

Updated slides will be availabe at http://wwwcg.in.tum.de/Tutorials/PacificVis09

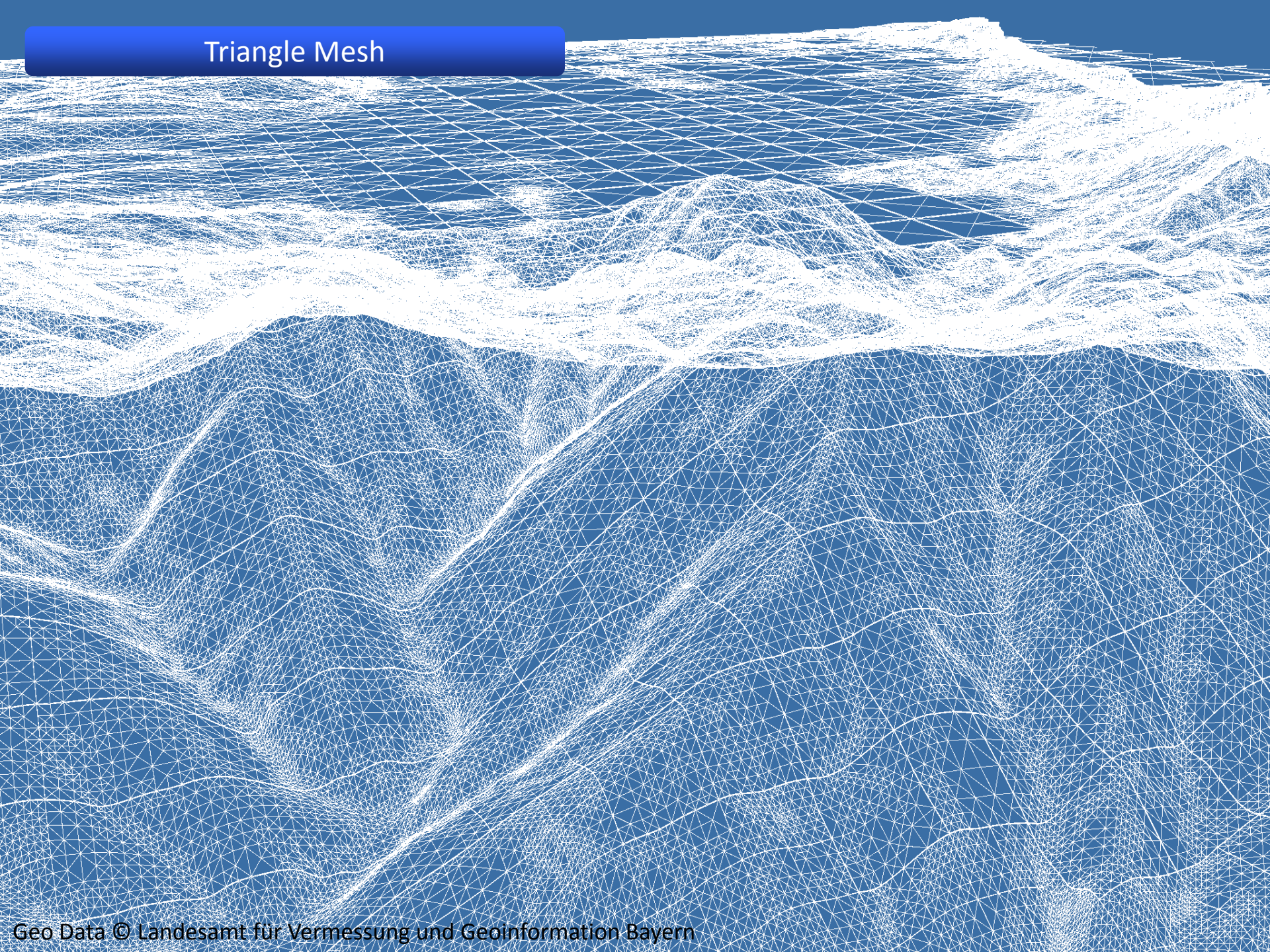
Terrain Data

Orthophoto

Height Field (DEM)

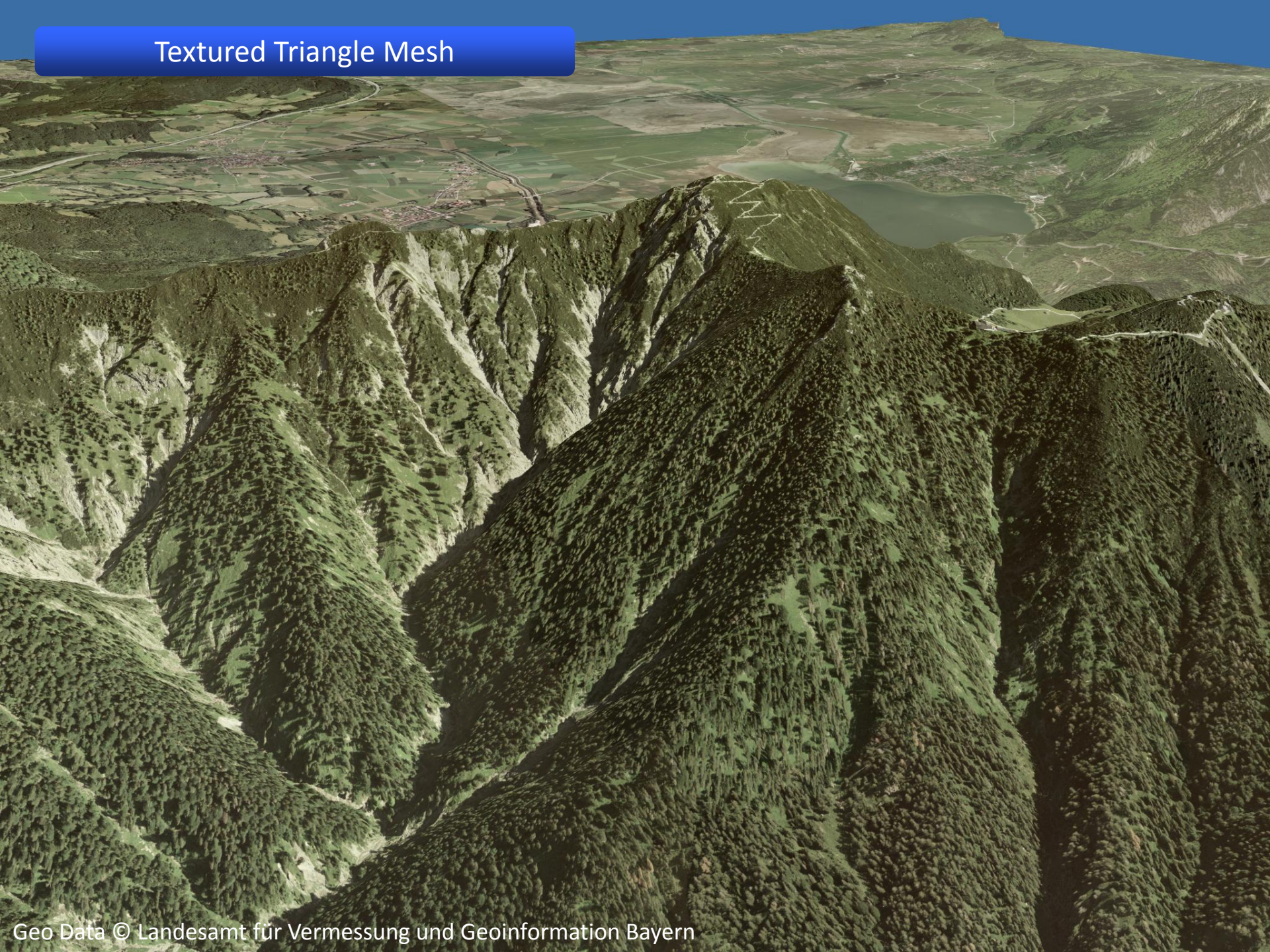


Triangle Mesh



Geo Data © Landesamt für Vermessung und Geoinformation Bayern

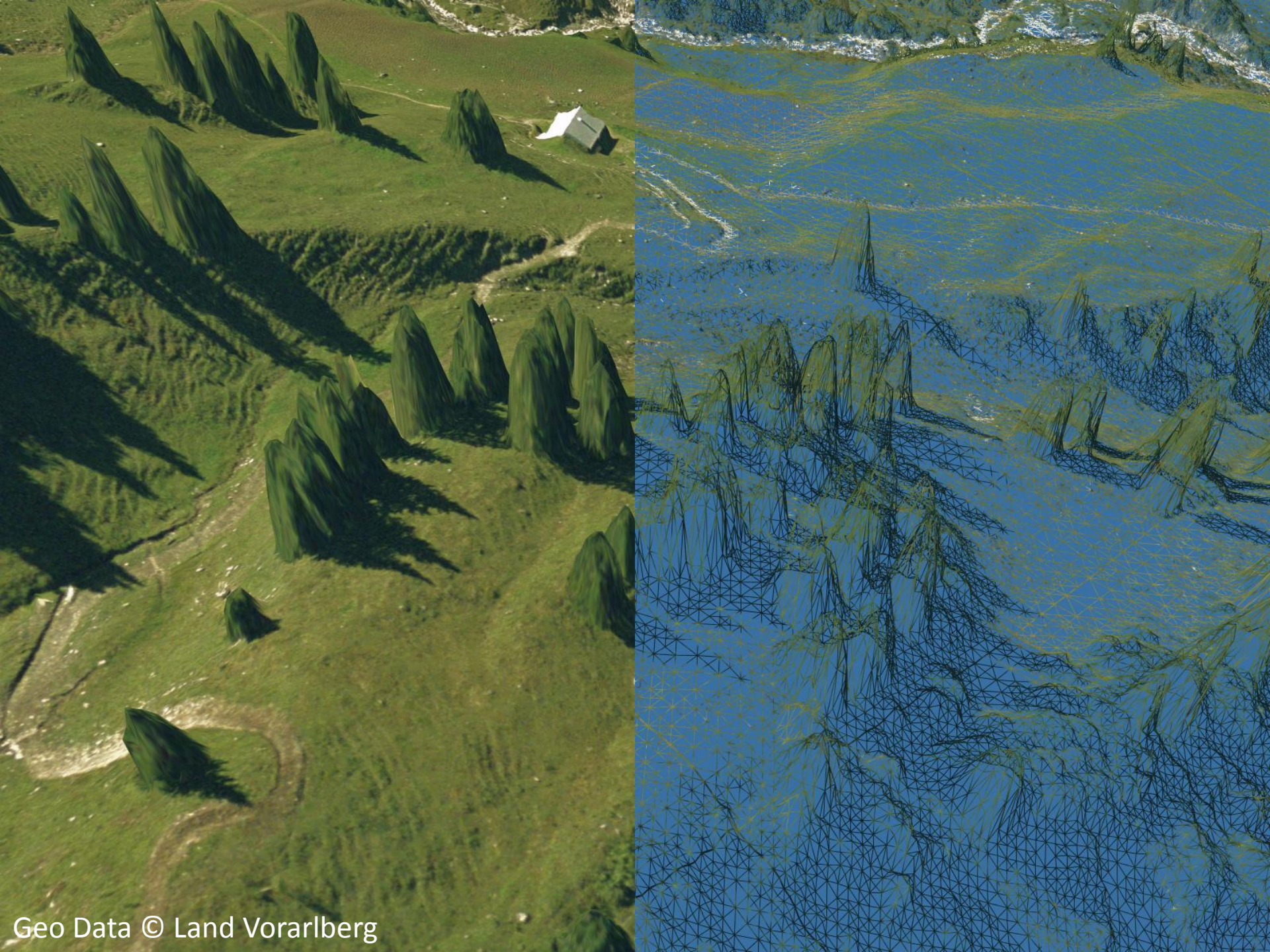
Textured Triangle Mesh

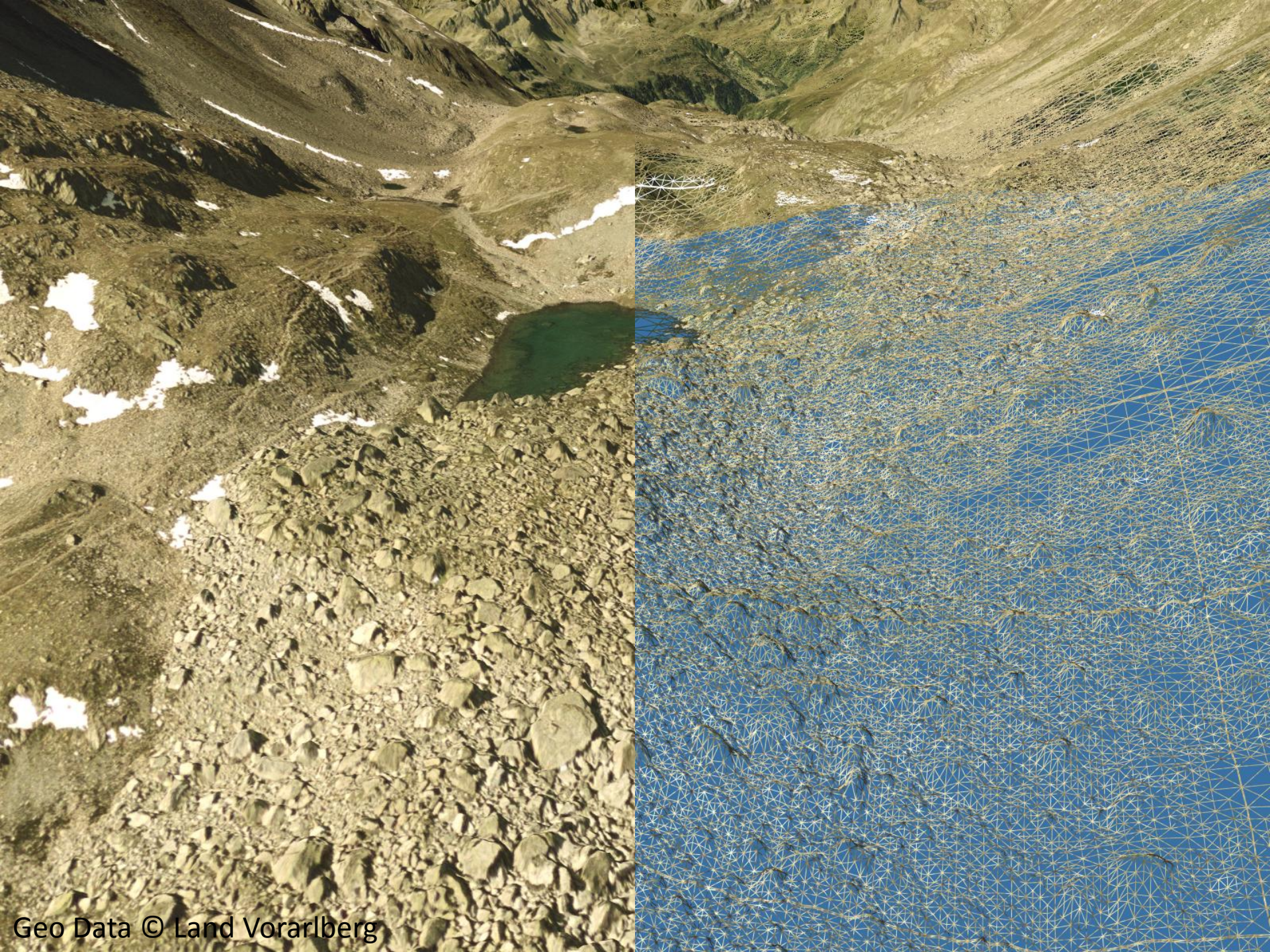


State of Utah, USA

Texture: 1m
Height Field: 5m
Extent: 460km x 600km
Raw Data: 790 GB
(China or USA: 27 TB)


Vorarlberg, Austria
Texture: 12.5cm
Height Field: 1m
Extent: 56km x 85km
Raw Data: 860 GB
(China or USA: 1690 TB)
Geo Data © Land Vorarlberg


Geo Data © Land Vorarlberg


Geo Data © Land Vorarlberg

Terrain Rendering – Application Areas

Games

- High, constant frame rates
- Quality advantageous but not prioritized

Simulators

- Constant frame rates (~30 fps)
- High degree of realism and detail, optionally stereo

Geographic Information Systems (GIS)

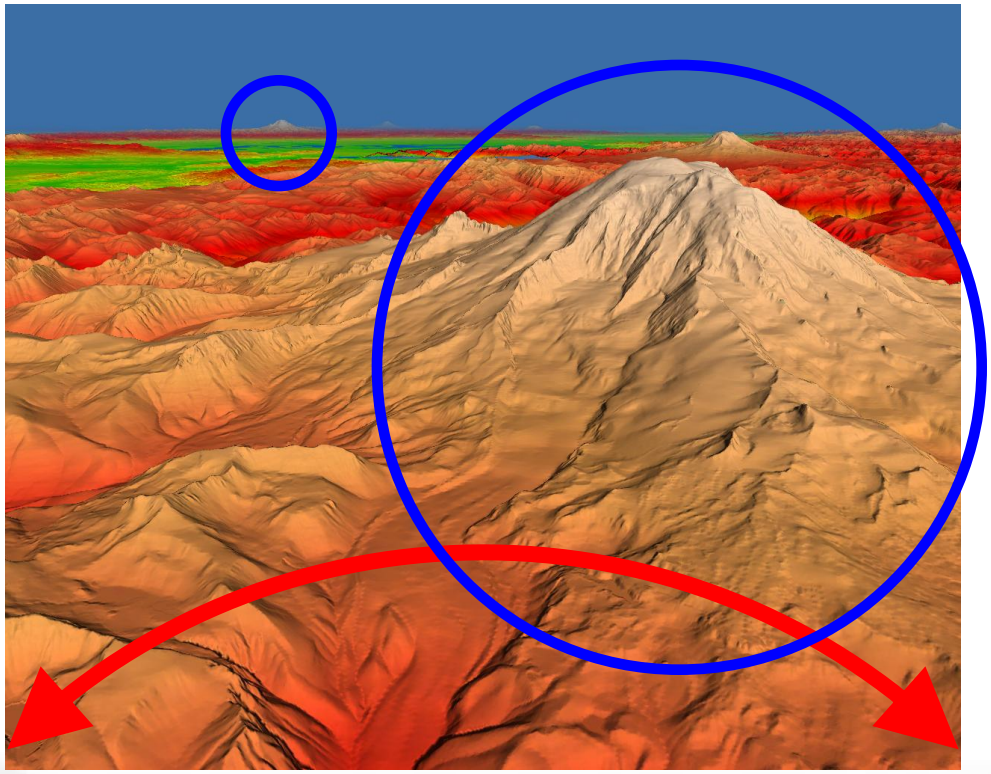
- Interactive frame rates advantageous (15+ fps)
- High resolution and precision

Terrain Rendering Principles

- How to render TB-sized datasets on standard PC hardware?
  - Limited memory bandwidths and capacities
  - Limited rendering throughput (~350 MΔ/s)
  - Brute force not possible …

Terrain Rendering Principles

- Only a small amount of data is visible per view



Perspective Projection

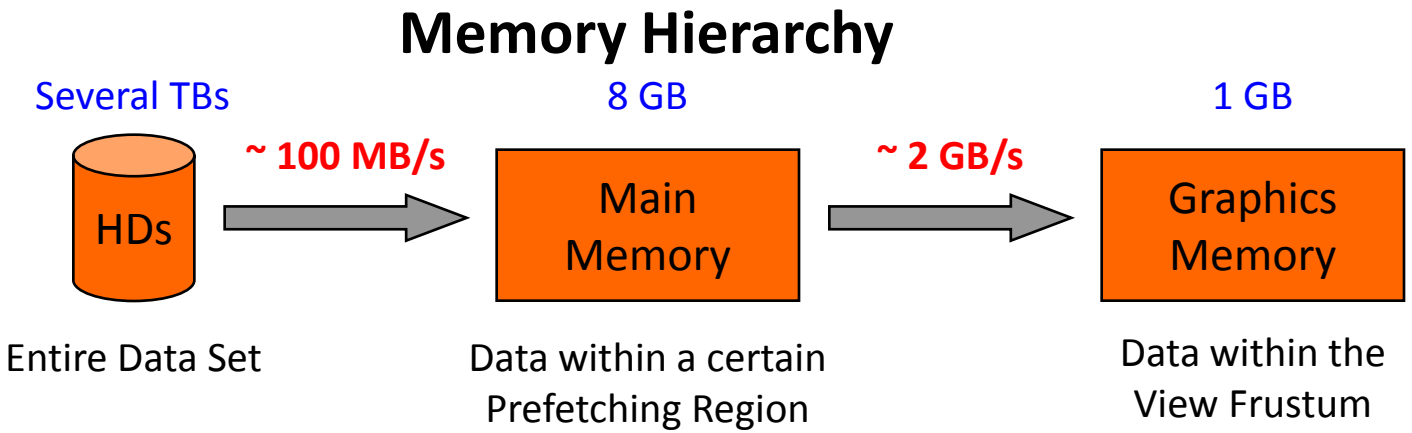
Level of Detail

Limited Field of View

View Frustum Culling

Terrain Rendering Principles

- Out-of-core data streaming



Hide Disk Access Latencies

Data Prefetching

Resource Allocation/Deallocation
Caching Strategies

CPU & GPU Memory Management

Reduce Memory Capacity
and Bandwidth Requirements

Data Compression

CLOD Terrain Rendering

Dynamic remeshing

- ROAMing Terrain: Real-time Optimally Adapting Meshes [Duchaineau et al., 1997]

Tile-based multiresolution mesh representation

- Rendering Massive Terrains using Chunked Level of Detail Control [Ulrich, 2002]

Nested regular grids

- Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids [Losasso and Hoppe, 2004]

Ray-casting of the height field

- GPU Ray-Casting for Scalable Terrain Rendering [Dick et al., 2009]

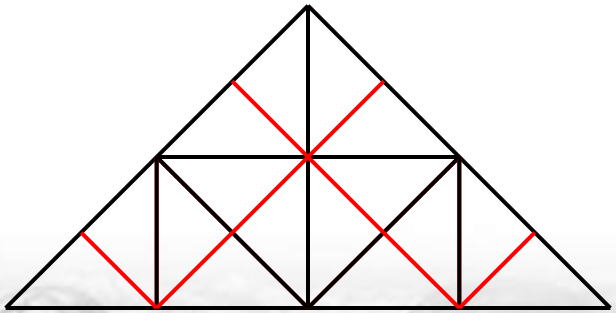
See www.vterrain.org for a list of publications …

Dynamic Remeshing

- View-dependent, adaptive remeshing in every frame
- LOD computation and view frustum culling per triangle
- Remeshing is done on the CPU, mesh has to be transferred to the GPU in every frame
- **ROAM: R**eal-time **O**ptimally **A**dapting **M**eshes

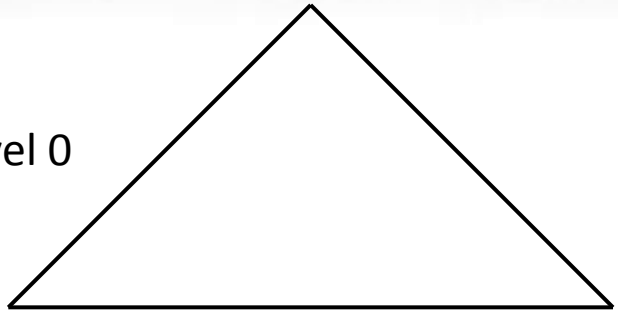
Dynamic Remeshing – ROAM

- ROAM uses a Triangle Bintree Mesh for view-dependent, adaptive meshing
  - Mesh consists of isosceles right triangles
  - Starting from a single triangle, triangles are successively split from the apex to the midpoint of the hypotenuse (longest edge bisection)

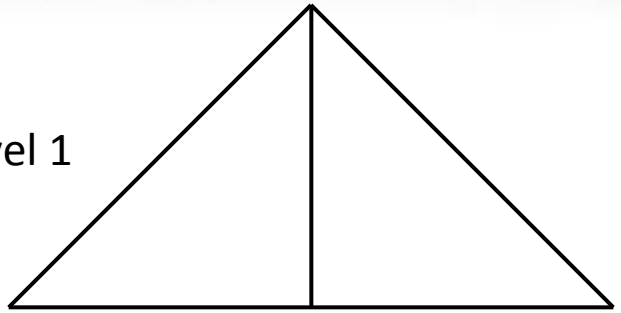


Dynamic Remeshing – ROAM

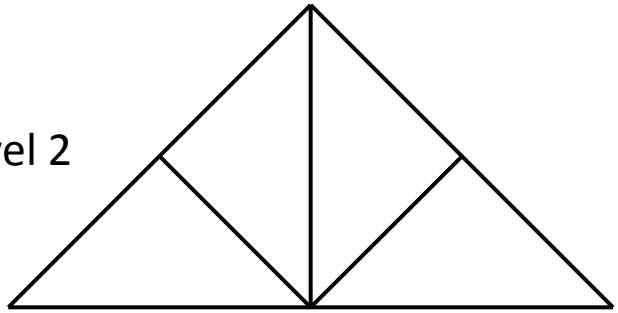
Level 0



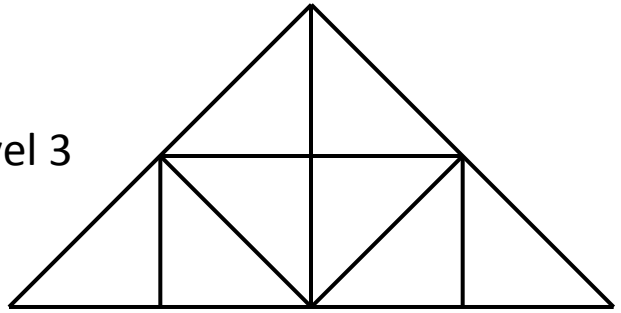
Level 1



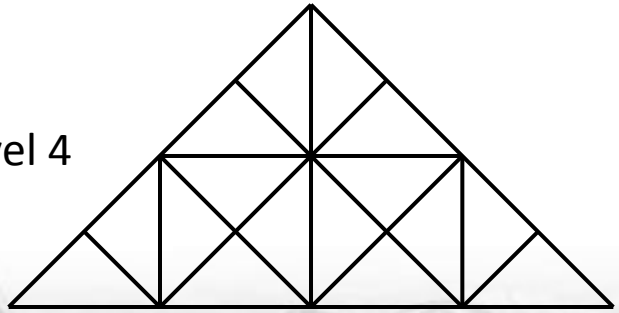
Level 2



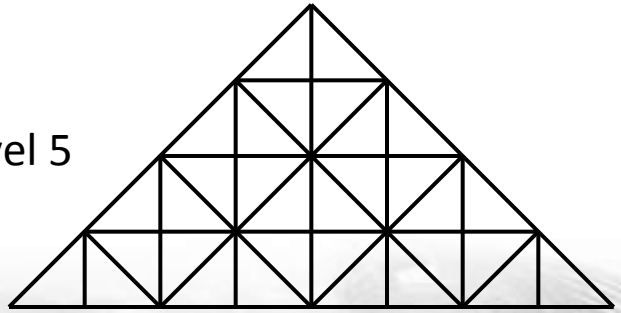
Level 3



Level 4

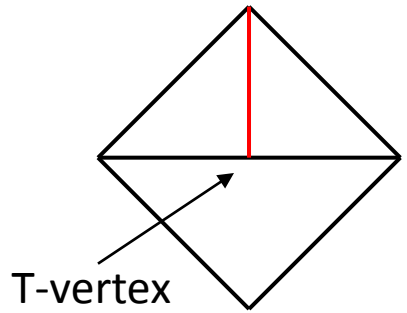


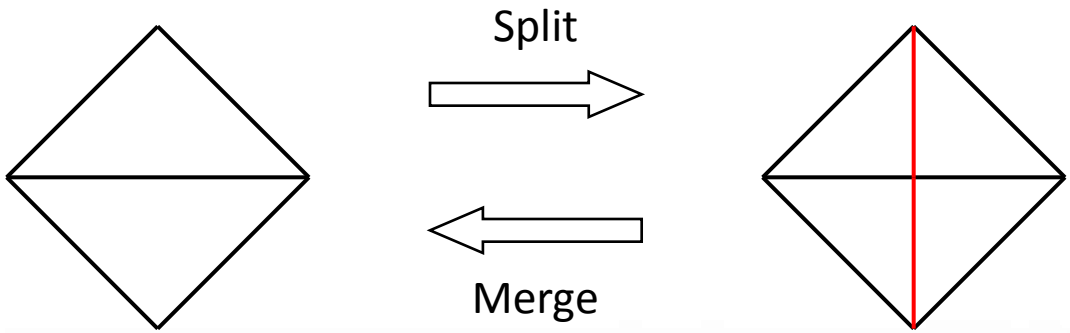
Level 5



Dynamic Remeshing – ROAM

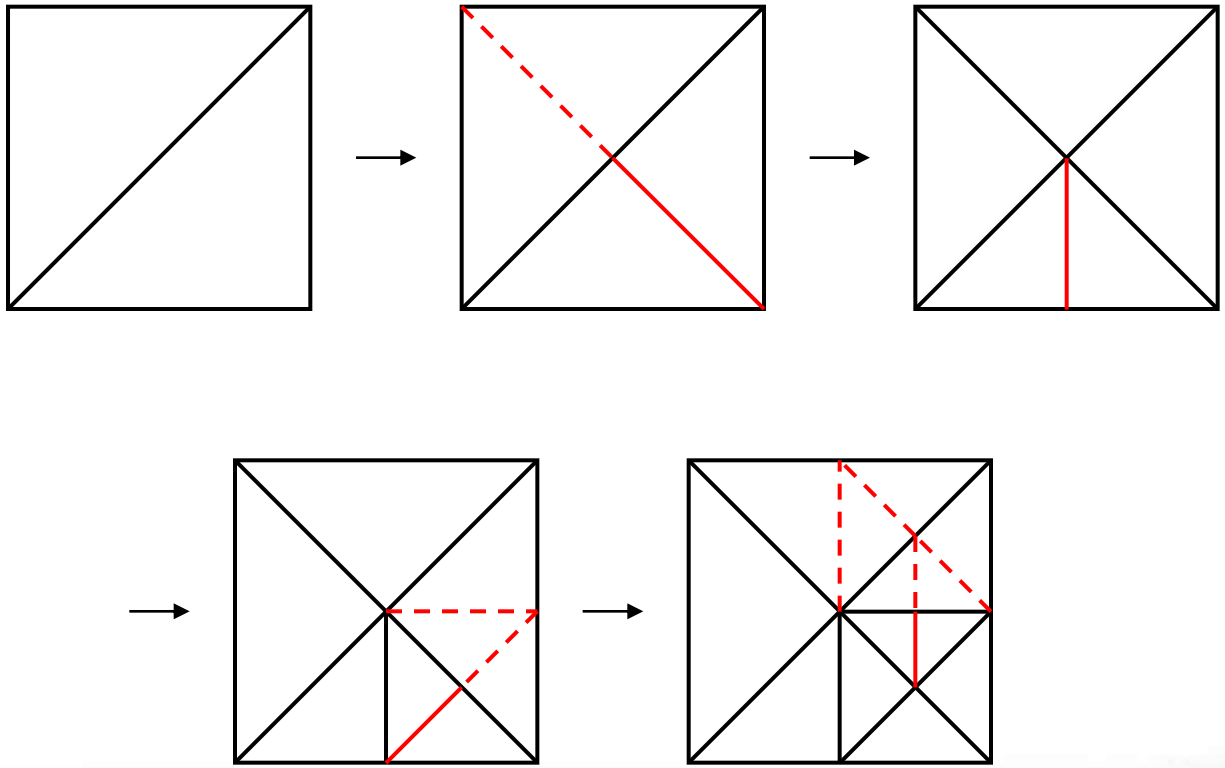
- Remeshing is done by exploiting frame-to-frame coherence
  - **Mesh refinement:** Split triangles using diamond splits to avoid T-vertices
  - **Mesh coarsening:** Merge diamonds





Dynamic Remeshing – ROAM

- Diamond splitting rule leads to forced splits

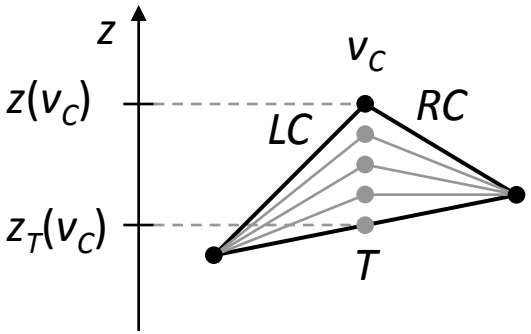


Dynamic Remeshing – ROAM

- Remeshing is driven by two priority queues
  - Priority = Screen space error
  - Split Queue: Force-split triangles with highest priority
  - Merge Queue: Merge diamonds with lowest priority
  - Uses *view-independent,* precomputed world-space error bounds for the triangles in the bintree
  - *View-dependent* priorities are obtained from these error bounds by projection into screen-space
    (Priorities have to be updated when view position changes)

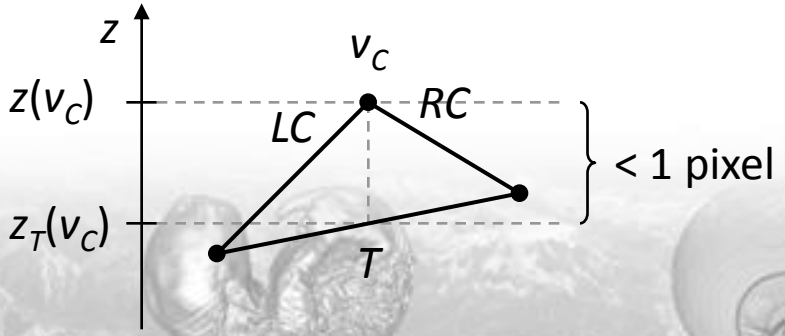
Avoiding Popping Artifacts

- Geomorphs (vertex morphing)
  - Linear interpolation of the height of the center vertex



$$z(t) = (1-t)\,z_T(v_C) + t\,z(v_C),$$
$$t \in [0, 1]$$

- Subpixel geometric screen-space error
  - Use screen space error tolerance < 1 pixel



Dynamic Remeshing

- Advantages:
  - Continuous triangulation (no T-vertices, no cracks)
  - Represents terrain using a minimum number of triangles
- Disadvantages:
  - High CPU load
  - High load on the CPU-GPU bus

Tile-based Terrain Rendering

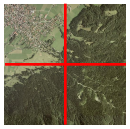
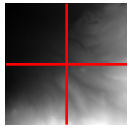
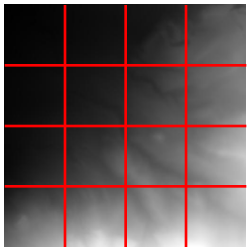
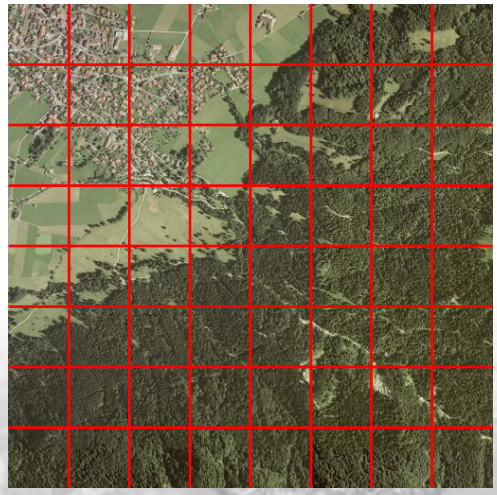
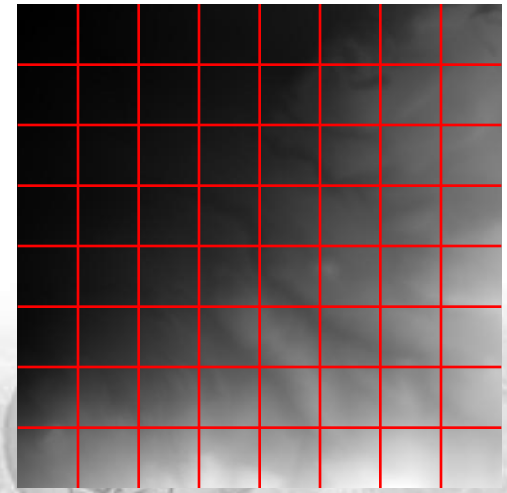
- Terrain is represented at multiple levels of details
- Each level is divided into tiles, which are organized as a quadtree
- Each tile is meshed separately in an offline preprocess (with respect to a prescribed world-space error tolerance per LOD)

Tile-based Terrain Rendering

- In each frame, the set of tiles is determined which represents the terrain at a prescribed screen-space error, using LOD computation and view frustum culling (per tile)
- Only new tiles are uploaded to the GPU (exploiting frame-to-frame coherence)
- **Chunked LOD:** Rendering Massive Terrains using Chunked Level of Detail Control [Ulrich, 2002]
  - Demo with source code available at http://tulrich.com/geekstuff/chunklod.html

Tile-based Terrain Rendering

- Multi-resolution, tile-based terrain representation

Level 3		1x1 Tile	 512^2 Samples
Level 2		2x2 Tiles	 1024^2 Samples
Level 1		4x4 Tiles	 2048^2 Samples
Level 0		8x8 Tiles	 4096^2 Samples

Tile-based Terrain Rendering

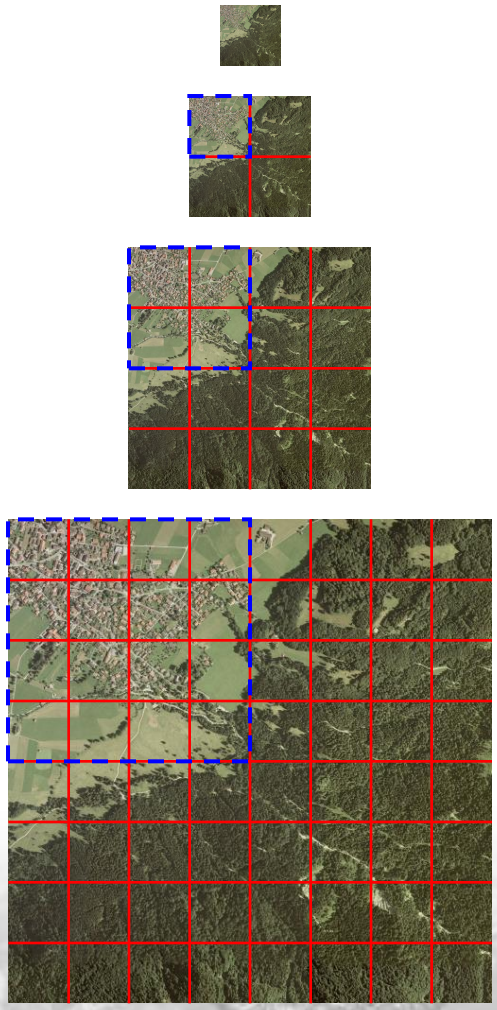
- Facts about the Tile Quadtree

Level 3
1x1 Tile

Level 2
2x2 Tiles

Level 1
4x4 Tiles

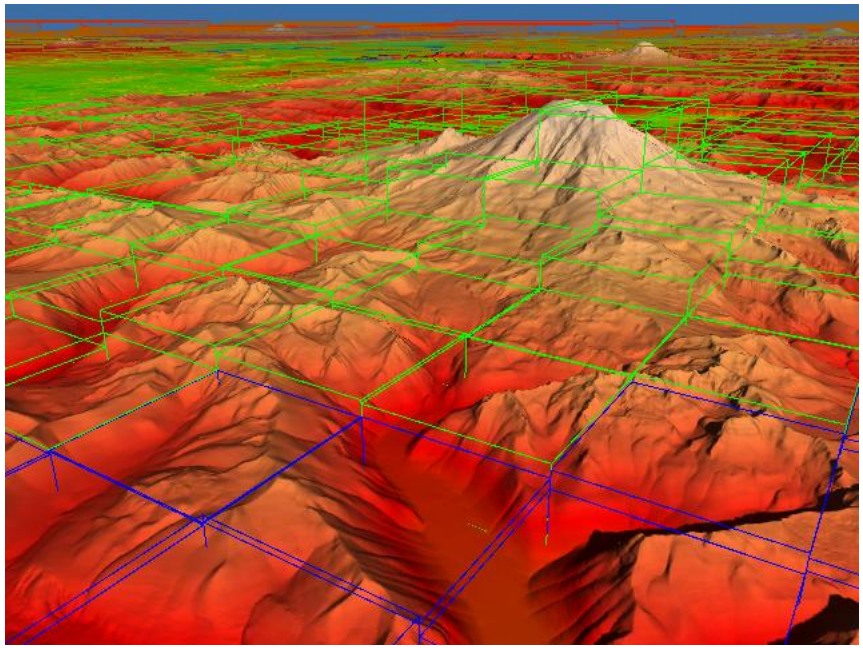
Level 0
8x8 Tiles

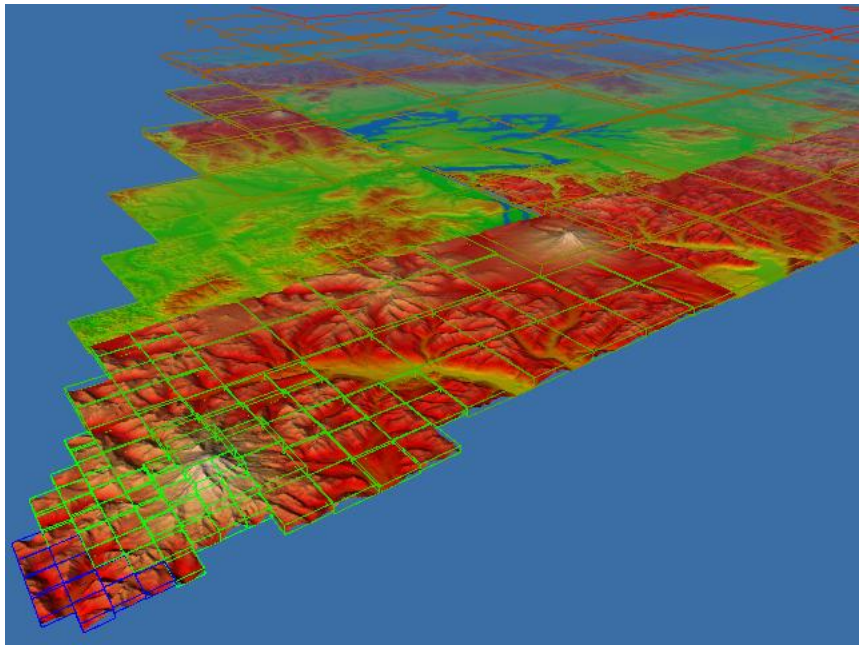


- 512^2 samples per tile
- World-space sample spacing ε_ℓ and world-space tile extent 512 ε_ℓ are doubled with each level ℓ (bottom-up)
- The tiles are meshed with a world-space error tolerance of ε_ℓ

Tile-based Terrain Rendering

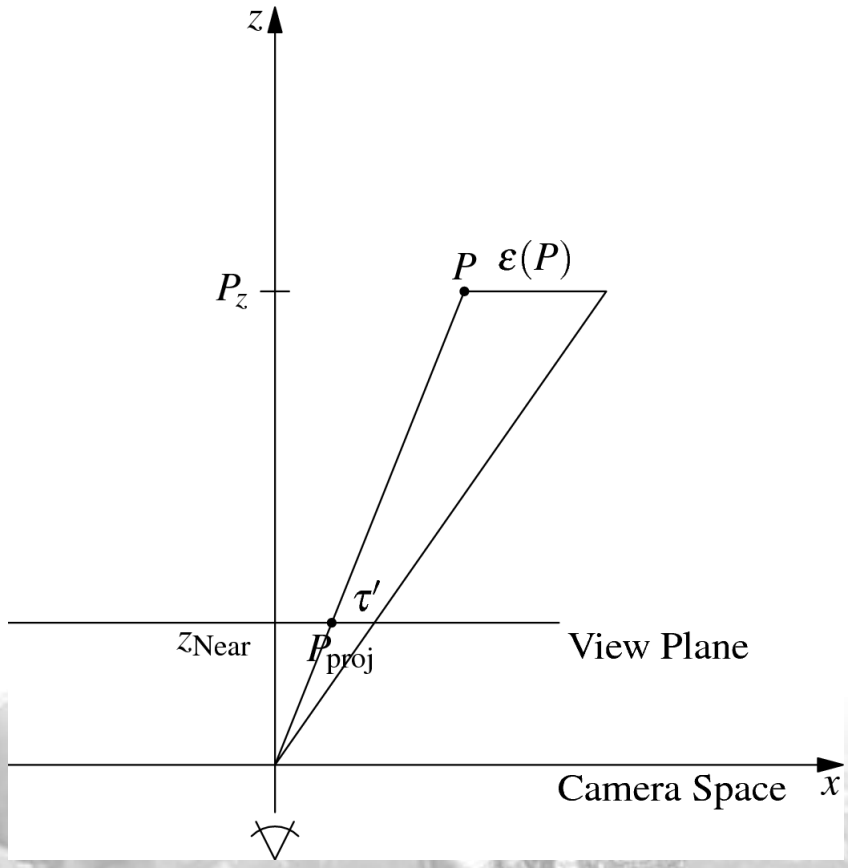
- View frustum culling and LOD computation per tile





Tile-based Terrain Rendering

- LOD computation is based on the theorem on intersecting lines



$$\frac{\varepsilon(P)}{P_z} = \frac{\tau'}{z_{\text{Near}}}$$

$$\varepsilon(P) = \frac{2}{h} \cdot \tan\left(\frac{\theta}{2}\right) \cdot \tau \cdot P_z$$

$\varepsilon(P)$: World-space error tolerance at point P

h: View port height in pixels

θ: fovy

τ: Screen-space error tolerance in pixels

P_z: Camera-space depth of point P

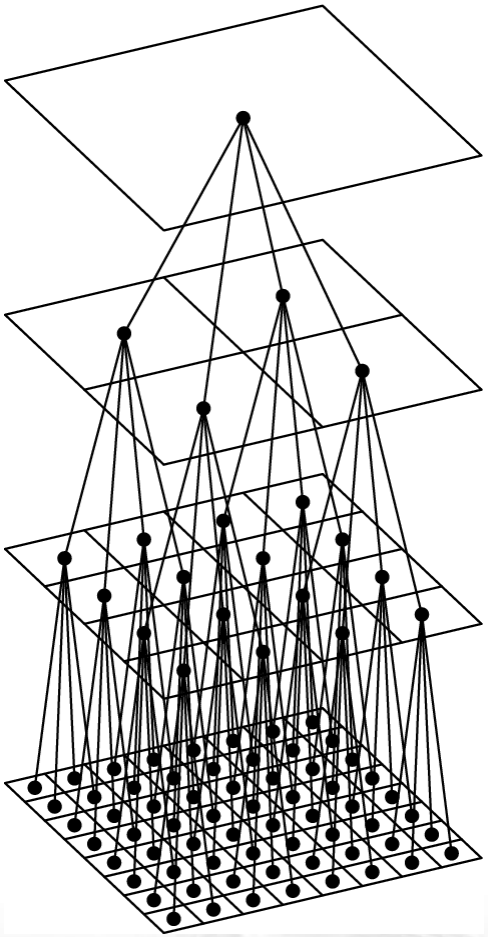
Use a tile at level ℓ for rendering, iff

$$\varepsilon_\ell \le \varepsilon(P') < \varepsilon_{\ell+1}$$

where P' denotes the point of the tile's bbox with the least depth to the camera

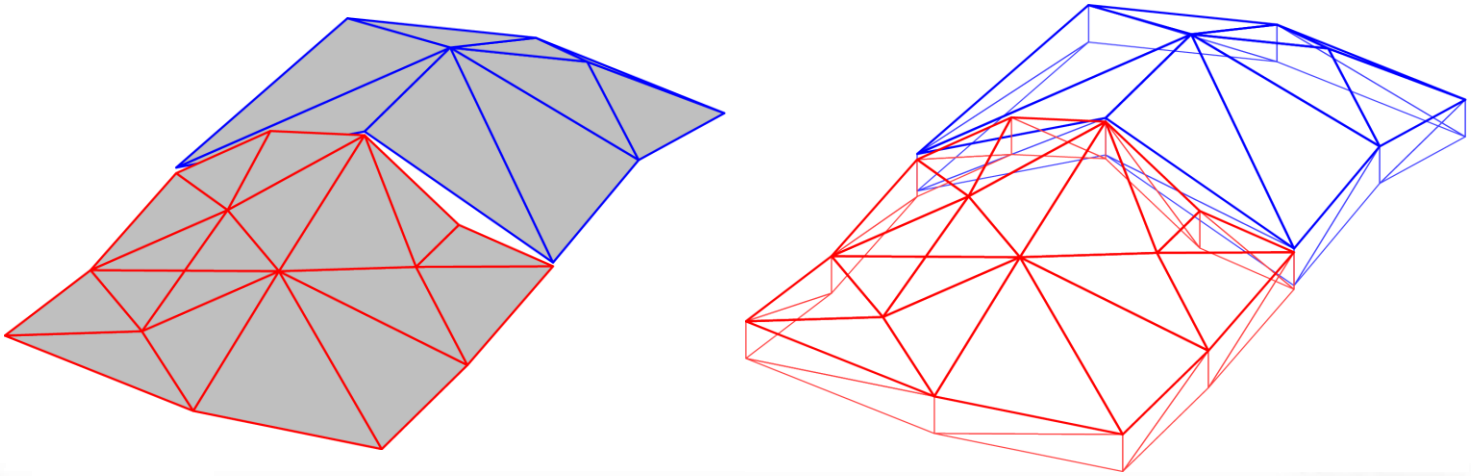
Tile Selection Algorithm

- To determine the set of tiles to be rendered, traverse the tile quadtree in preorder
  - If a tile is culled, skip all descendants
  - If a tile is not culled and its LOD is sufficient with respect to the prescribed screen-space error tolerance, add this tile to the set and skip all descendants



Fixing Cracks between Tiles

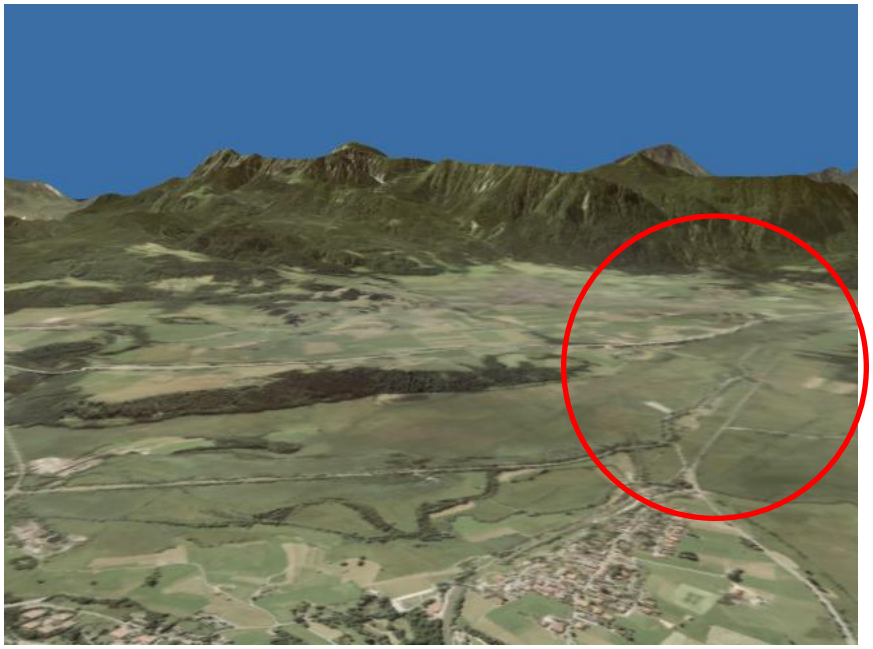
- Tiles are meshed independently, thus cracks (due to T-vertices and quantization errors) can occur at the tile boundaries
- Render skirts around each tile to hide cracks



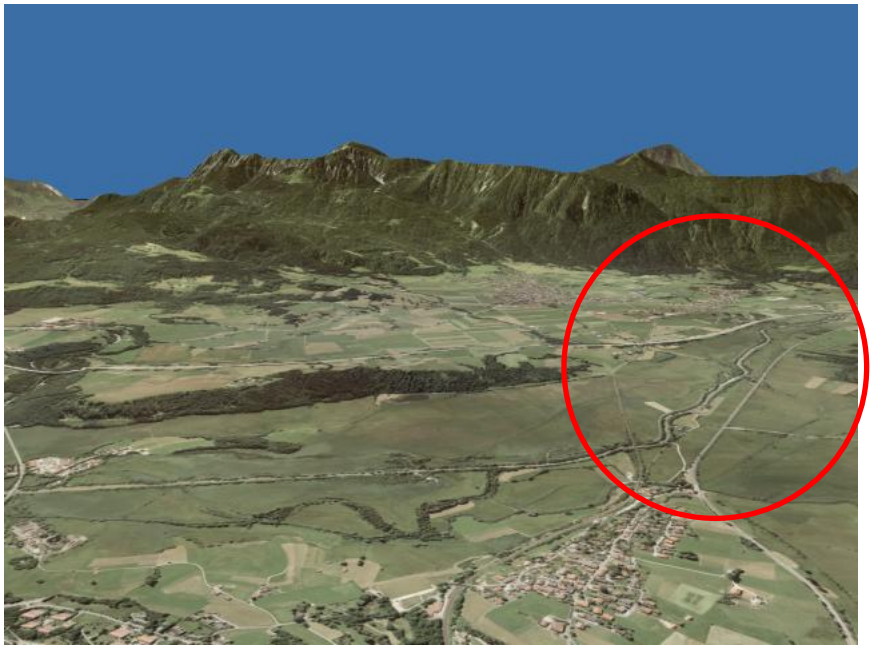
- Alternatives: Flanges, Zero-Area-Triangles (see [Ulrich, 2002])

Texturing

- Anisotropic texture filtering is mandatory for terrain rendering



Mipmapping



Anisotropic Filtering

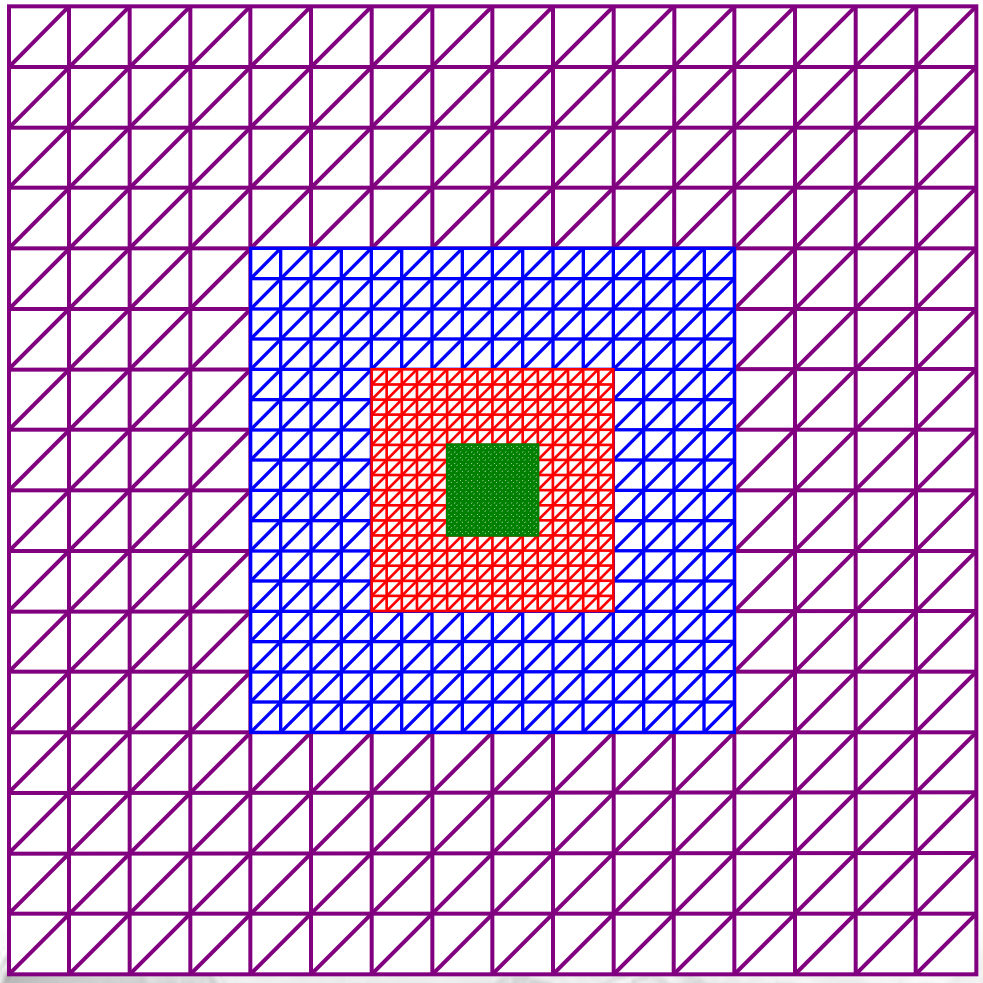
Tile-based Terrain Rendering

- Advantages:
  - Low CPU load (computations are done per tile)
  - Moderate load on the CPU-GPU bus due to frame-to-frame coherence
- Disadvantages:
  - Cracks between tiles
  - Higher number of triangles than necessary

Nested Regular Grids

- **Geometry Clipmaps:** Terrain Rendering Using Nested Regular Grids
  [Losasso and Hoppe, 2004]
  - Demo available at http://research.microsoft.com/en-us/um/people/hoppe/
- Use a set of nested regular grids centered about the viewer

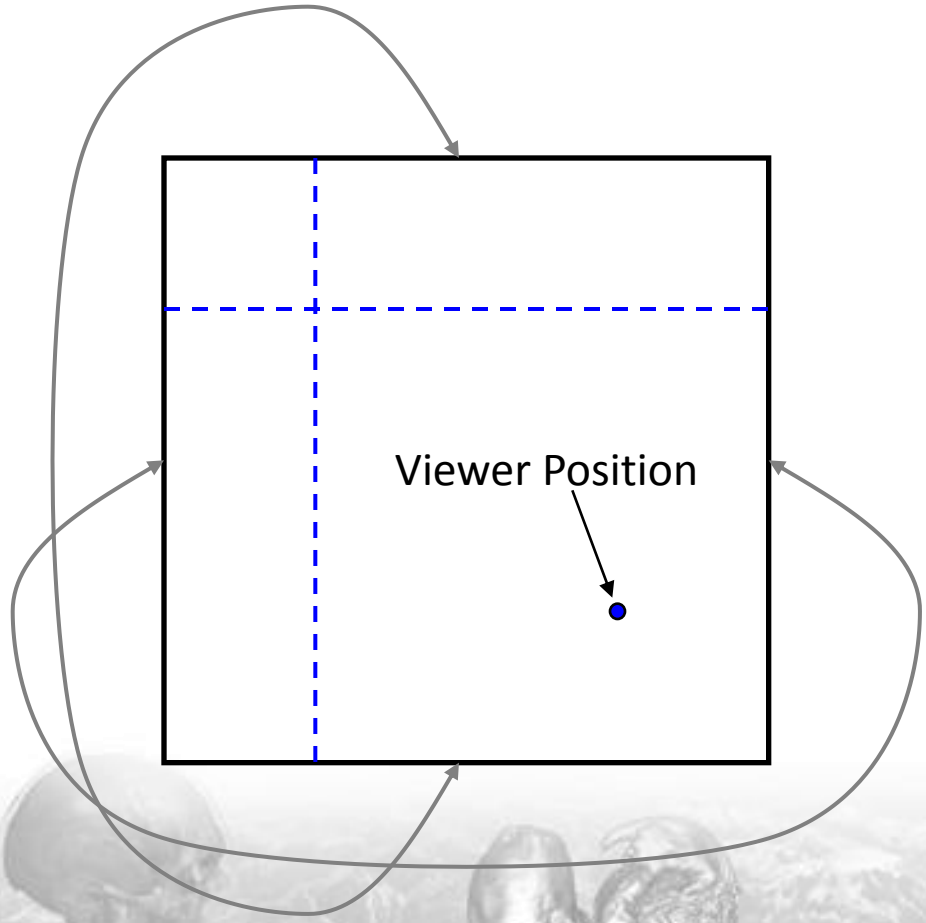
Nested Regular Grids

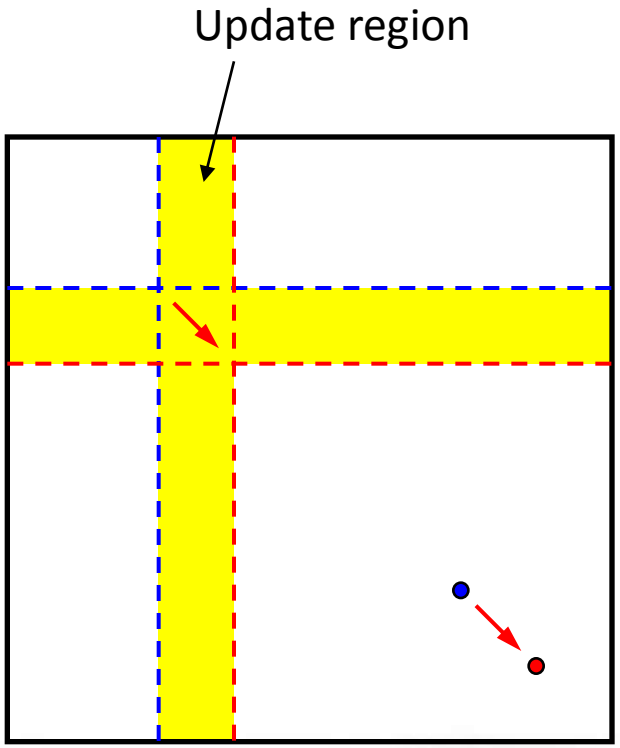


- Height values are fetched on-the-fly from a height field pyramid (the clipmap)
- LOD is controlled by the spatial extent of each grid
- View frustum culling is realized by dividing each grid into blocks

Nested Regular Grids

- Update of the clipmap based on toroidal access



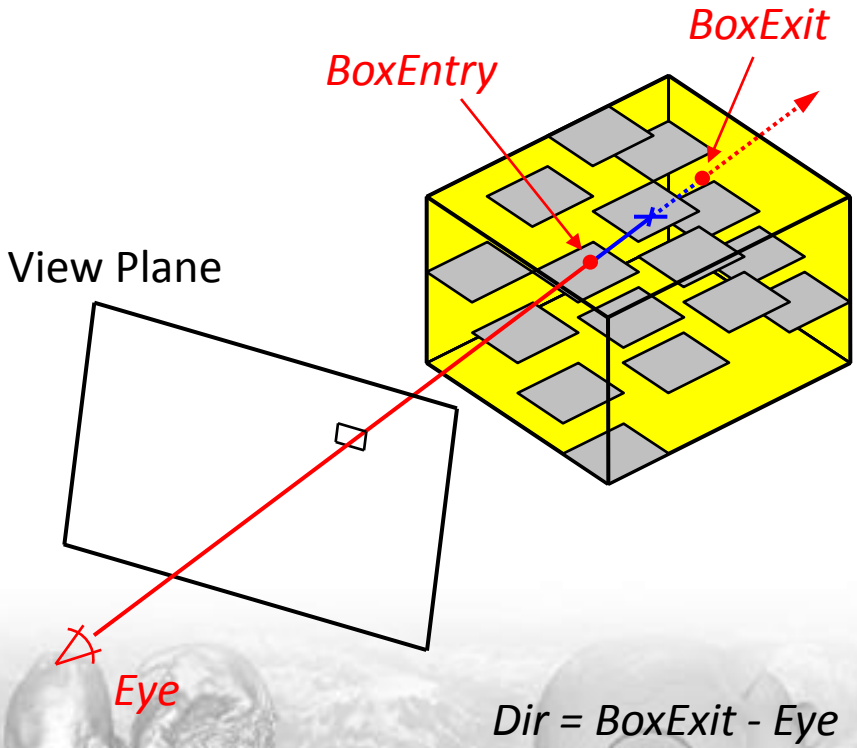


Nested Regular Grids

- Advantages:
  - Height field can be compressed using image compression methods
  - Simple memory management
- Disadvantages:
  - No exact screen space error control
  - Extremely high number of triangles

GPU-based Terrain Ray-Casting

- Ray-casting of the terrain height field
- GPU Ray-Casting for Scalable Terrain Rendering [Dick et al., 2009]



GPU-based Terrain Ray-Casting

- Advantages:
  - Performance fully independent of the complexity of terrain
  - Higher performance and lower GPU memory consumption than triangle-based rendering for *high-resolution* height fields
- Disadvantages:
  - For *coarse-resolution* height fields, triangle-based rendering is faster and requires less GPU memory

Data Compression

- Benefits:
  - Reduces memory capacity requirements
  - Reduces bandwidth requirements
- Favor schemes that can be decoded on the GPU
  - Reduces CPU load
  - Reduces CPU-GPU traffic
- Encoding generally not time-critical
  - Performed in a (time-consuming) preprocess

Texture Compression – S3TC

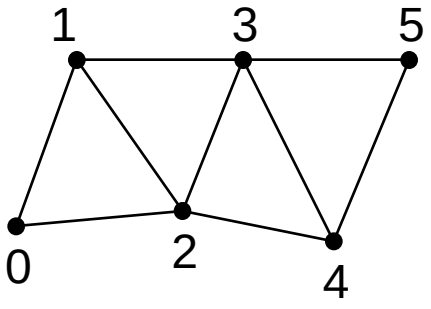
- **S3 T**exture **C**ompression, here: DXT1, no alpha [US Patent 6658146]
  - Asymmetric, lossy block truncation code
  - Standard compression scheme (DirectX, OpenGL)
  - GPU renders directly from compressed data
  - Divides textures into 4x4 blocks
  - Assigns a fixed rate of 64 bits per block (4 bpp)
  - Compression ratio 6:1 (R8G8B8)
  - Use the Squish library by Simon Brown for compression; Available with source code at http://code.google.com/p/libsquish/

Geometry Compression

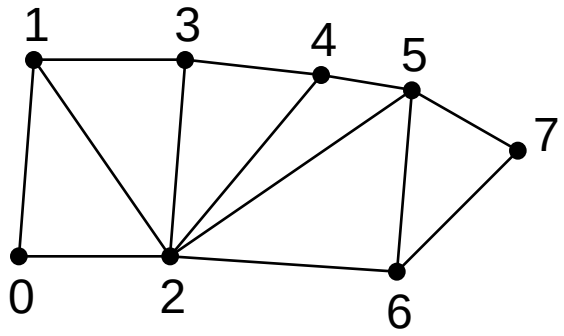
- Compression scheme for bintree meshes supporting GPU-based decoding
  - Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering [Dick et al., 2009]
  - Underlying 2D Mesh: Lossless compression based on a generalized triangle strip representation
  - Height values: Lossy compression based on uniform quantization
  - Compression rate 8-9 (wrt triangle list representation, 32 bits per vertex)

Geometry Compression

- Generalized Triangle Strip
  - Store only one vertex per triangle



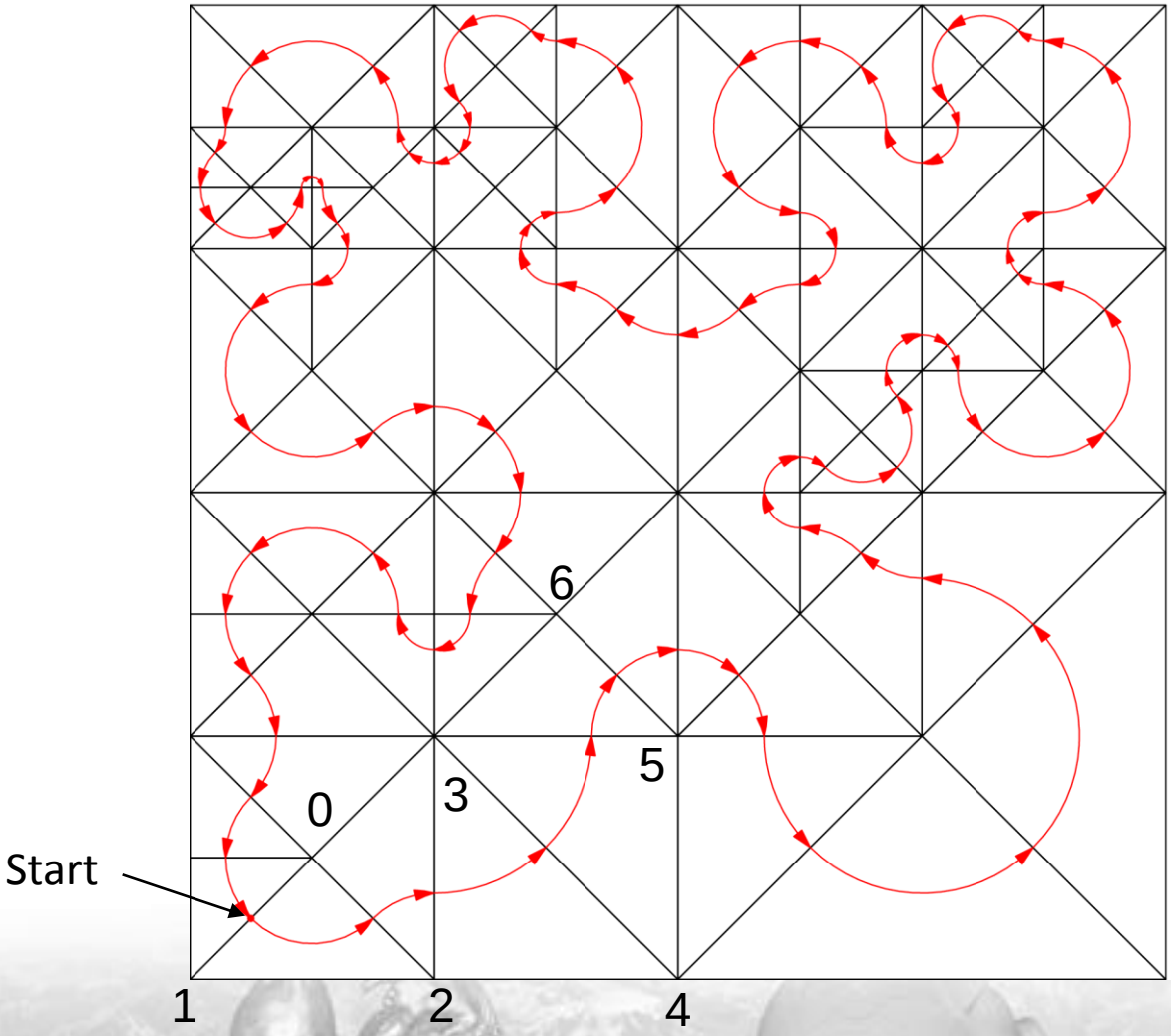
Regular Triangle Strip
(0-1-2, 2-1-3, 2-3-4, 4-3-5, …)



Generalized Triangle Strip

- Construct a directed path that
  - Enters each triangle exactly once
  - Leaves and enters triangles across edges
  - Hamiltonian path of dual graph

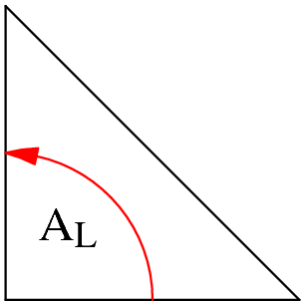
Geometry Compression

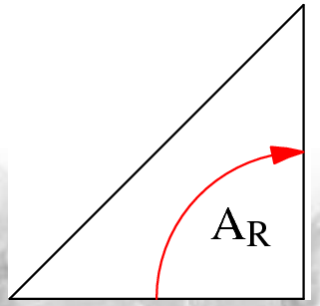


Geometry Compression

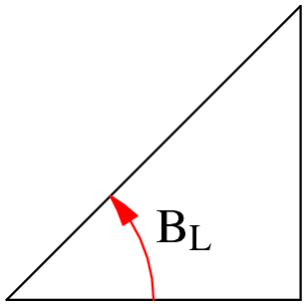
- Classify triangles by
  - Type of the entering/leaving edge (A, B, C)
  - Winding of the path (L, R)

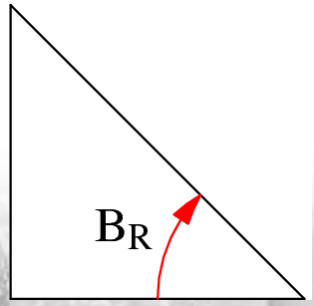
Type A:
Cathetus to cathetus



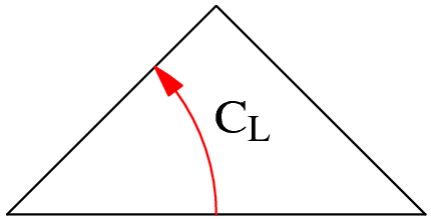


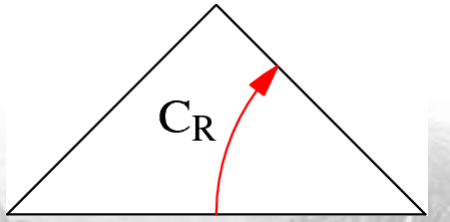
Type B:
Cathetus to hypotenuse





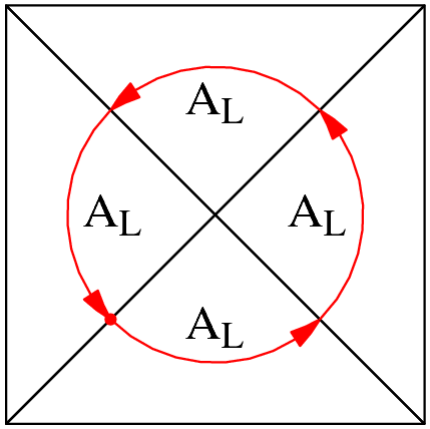
Type C:
Hypotenuse to cathetus





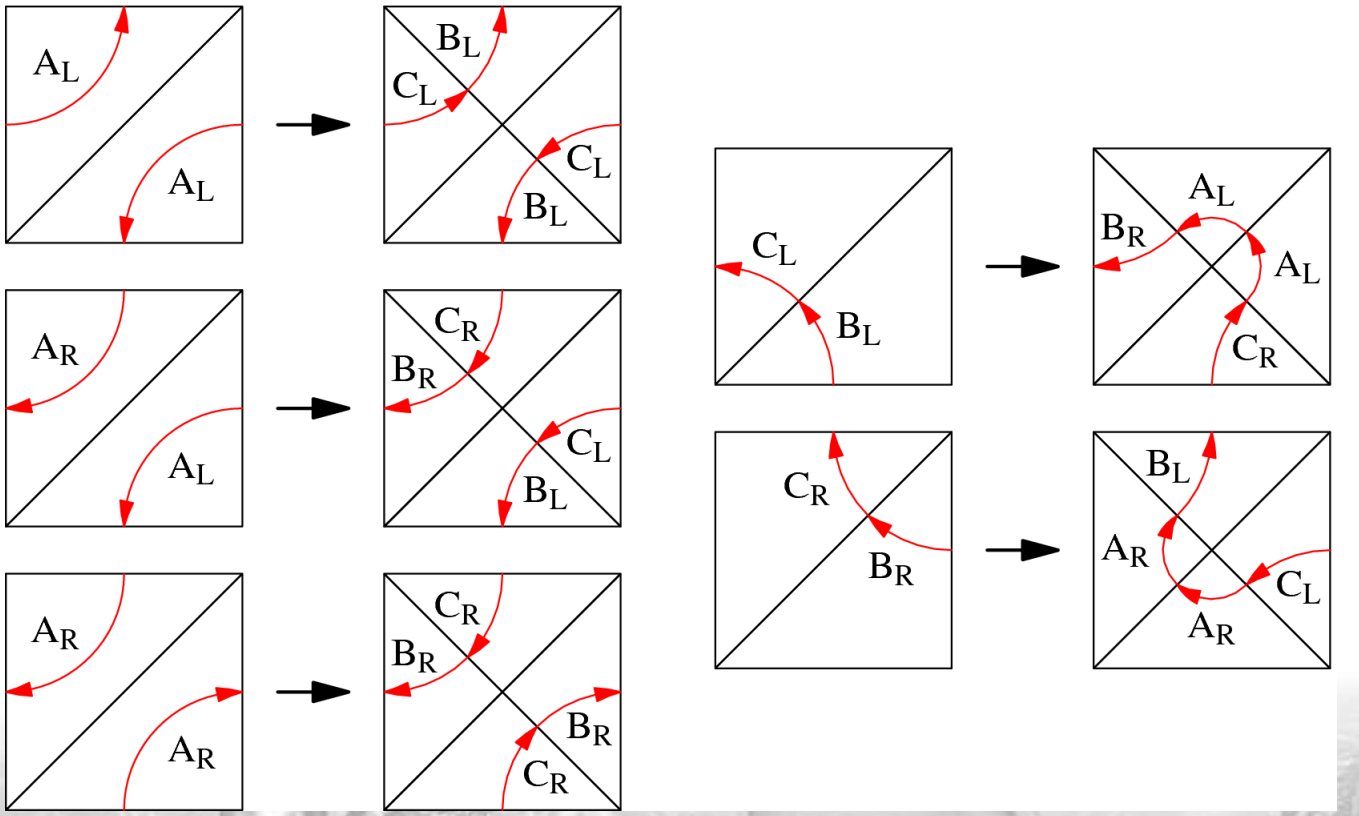
Geometry Compression

- Construct path during diamond splitting
  - Initial Mesh

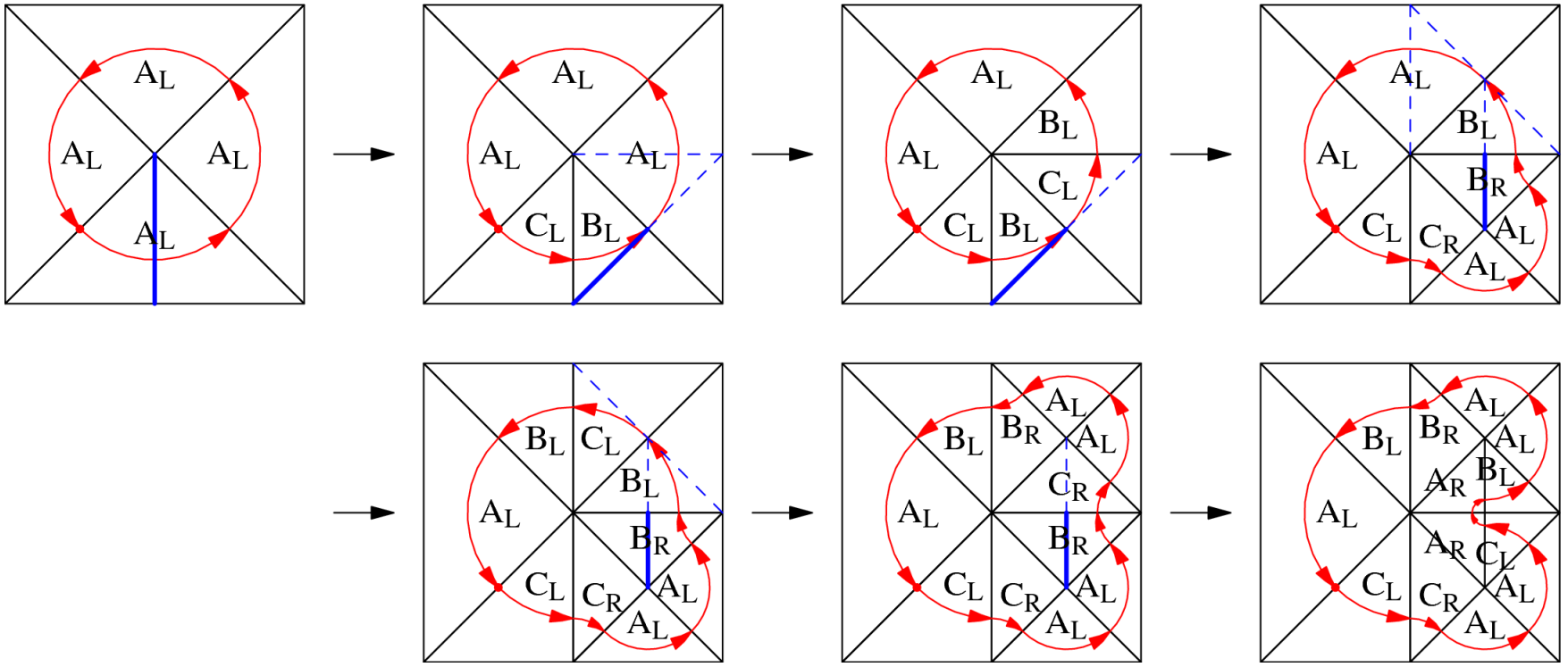


Geometry Compression

- Construct path during diamond splitting
  - Replacement System

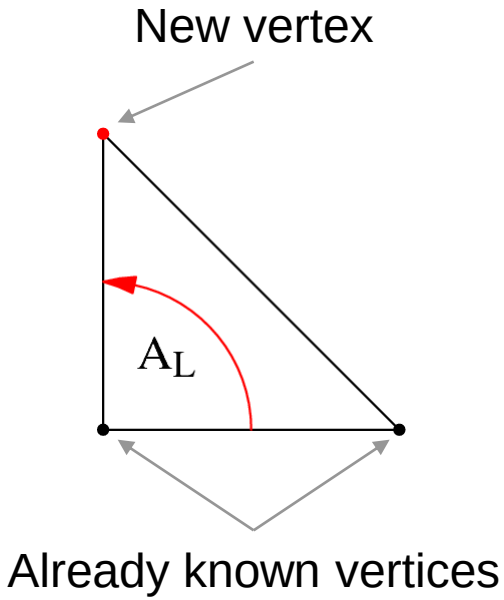


Geometry Compression



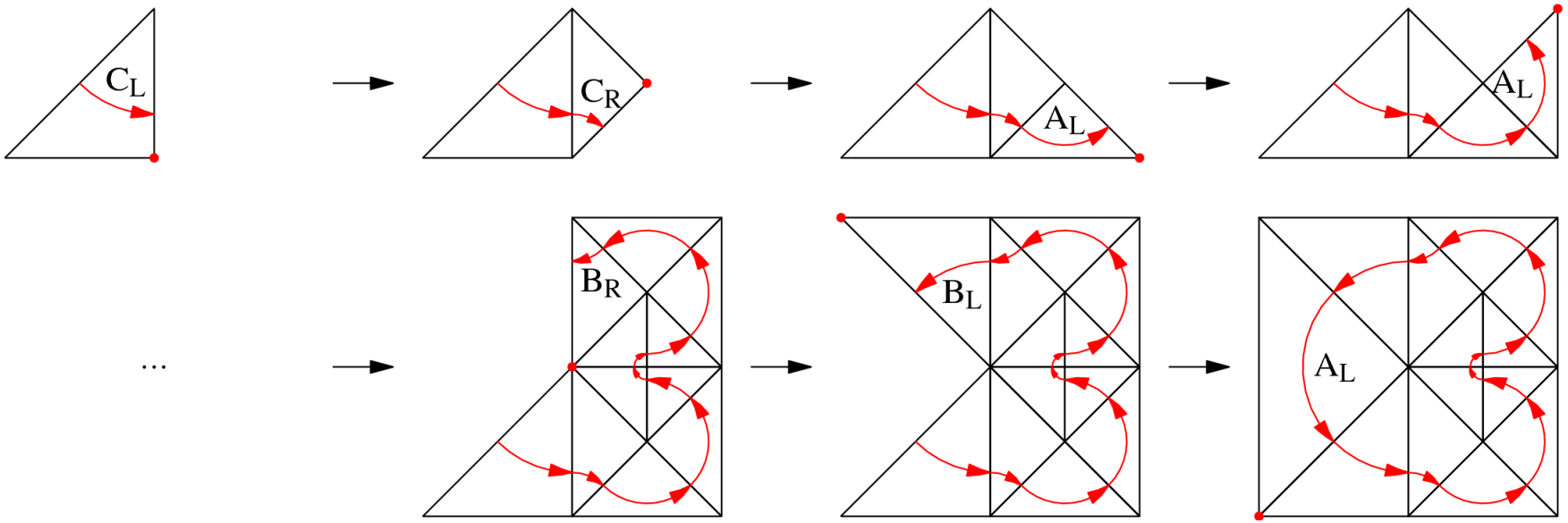
Geometry Compression – Encoding

- For each triangle
  - Store type (A,B,C)
  - Winding (L,R) can be inferred
  - Store height value of new vertex
- Bitrate
  - 2 bits for triangle type
  - Variable (per tile) #bits for height value



Geometry Compression – Decoding

- Encoded mesh:
  C_L, C_R, A_L, A_L, …, B_R, B_L, A_L + height values

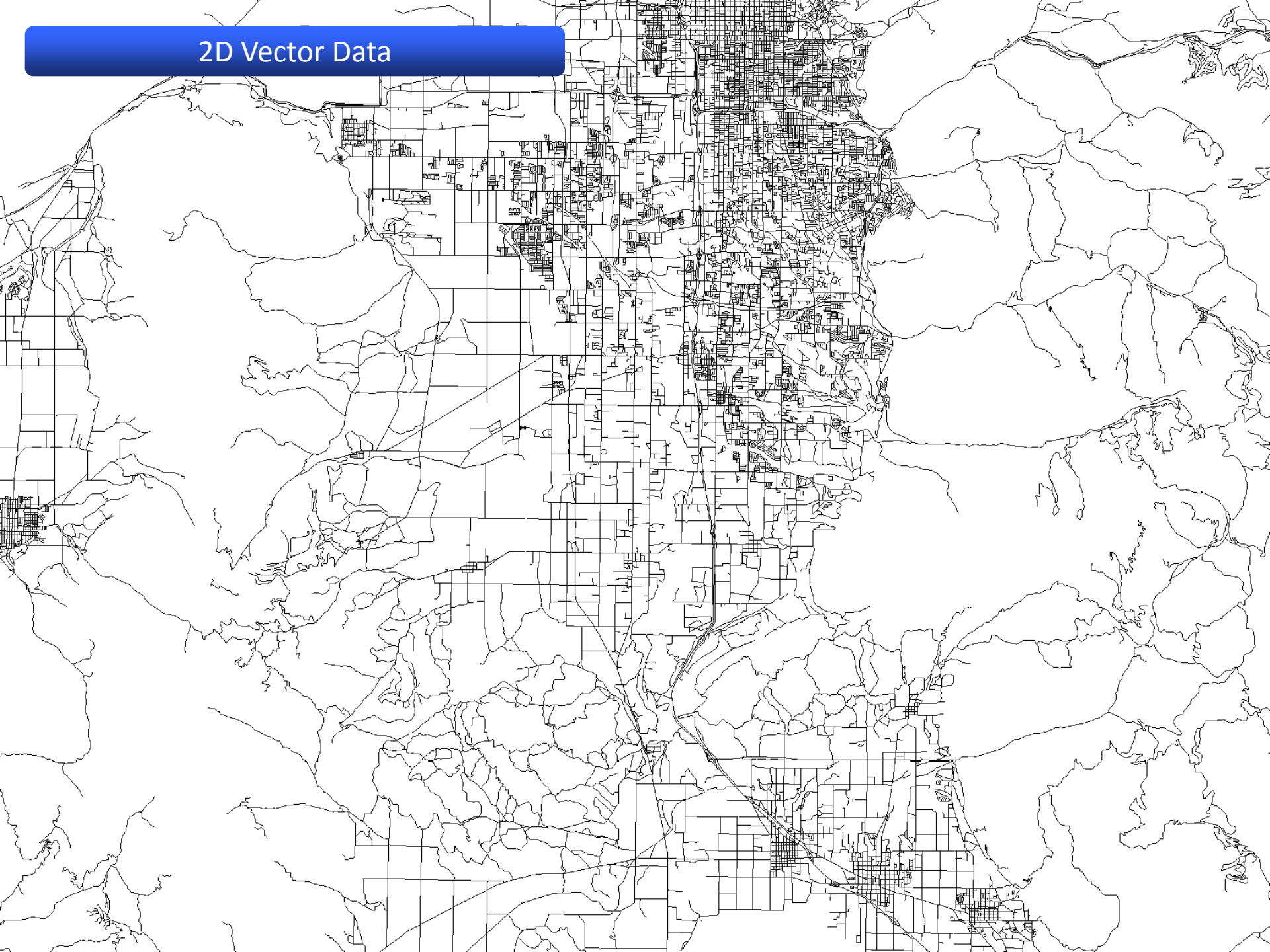


- Can be decoded directly on the GPU

Overlaying 2D Vector Data

- 2D Vector Data
  - Polyline and polygonal vector data
  - Roads, trails, villages, land use, …
- Overlaying onto the 3D terrain
  - For the visualization, the 2D vector data have to be mapped onto the 3D terrain surface …

2D Vector Data



3D Terrain


Vector data overlaid onto the terrain

Overlaying 2D Vector Data

Geometry-based

- Render vector data as 3D geometry (lines, triangles)
- Problems: Z-fighting, terrain LOD adaptation

Texture-based

- Rasterize vector data into texture, overlay texture
- Problems: Resolution, GPU memory consumption

“Shadow Volume” Approach

- Efficient and Accurate Rendering of Vector Data on Virtual Landscapes [Schneider and Klein, 2007]

Thanks for your attention!

Online Demo: Tile-based Terrain Rendering

- Data Set: State of Utah, USA
  - Texture / Geometry Resolution: 1m / 5m
  - Spatial Extent: 460km x 600km
  - Raw Data Volume: 790GB (Compressed 175GB)
- System: Notebook equipped with
  - Intel Mobile Core 2 Duo T7500, 2.2GHz
  - NVIDIA GeForce 8600M GS, 256MB video memory
  - 2GB of RAM
  - External Harddisk, connected via USB

Updated slides will be availabe at http://wwwcg.in.tum.de/Tutorials/PacificVis09