Efficient and Accurate Rendering of Vector Data on Large Virtual Landscapes

M. Schneider and R. Klein

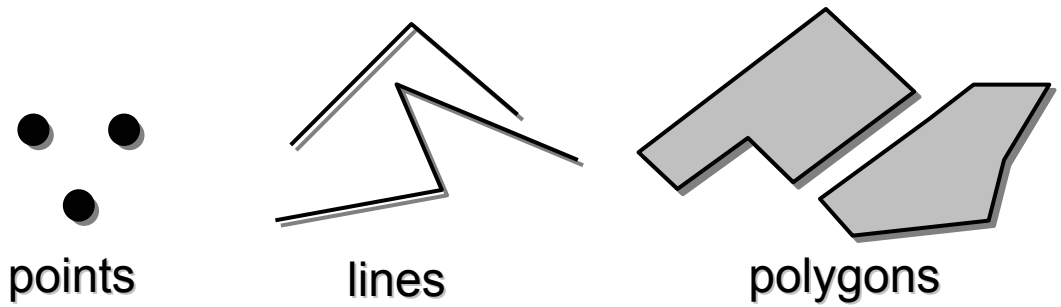
University of Bonn
Computer Graphics Group
www.cg.cs.uni-bonn.de

Motivation

- ever-higher sampled aerial photography and elevation data available
- efficient methods to render them at real-time framerates exist
- **But:** for most disciplines solely visualising the terrain is not enough
- need a way to define regions of interest and connect metadata with it

Motivation

- in the GIS domain so called **vector data** is used
- ordered set of georeferenced 2d points



- vector data has important applications in the analysis and management of virtual landscapes in many disciplines (geography, geology, archaeology, ...)

Question: How to render vector data on the terrain surface?

Previous work

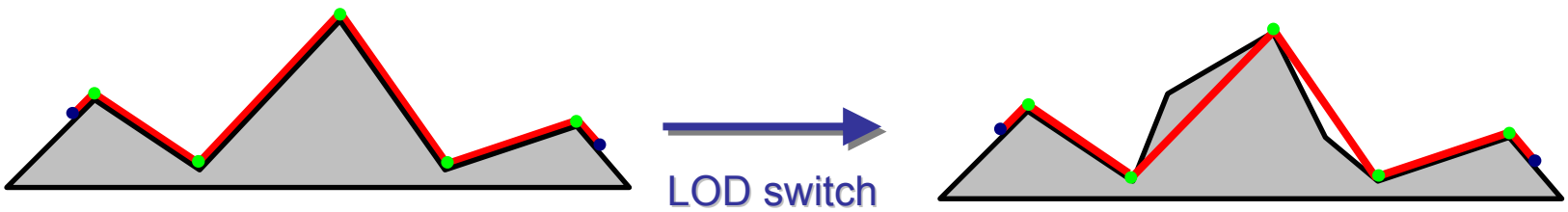
Two kinds of methods

- geometry-based
  - create 3d geometric primitives from the data
  - render vector geometry on top of terrain geometry
- texture-based
  - rasterize vector data into texture
  - project texture onto terrain surface

Geometry based approach

Problem:

- view-dependent LOD



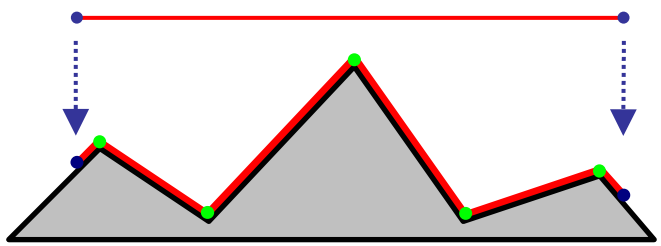
Possible solution:

- adapt vector data to current level-of-detail
  - at run-time for continuous LODs
  - as preprocessing for static LODs
    - create vector data geometry for each level-of-detail
    - incorporate vector data in quadtree

Geometry based approach

properties

+ high quality
- z-buffer stitching artifacts
- has to be adapted to the used rendering engine
- complexity of vector data visualization coupled with terrain complexity



Texture-based approach

naive texture-based approach

- rasterize vector data into texture in a preprocessing step
- use texture mapping to project texture onto terrain

properties

+ fast
+ independent of underlying geometry
- high texture memory requirements, in particular for multiple layers
- accuracy limited by texture resolution

Texture-based approach

Improvements:

- create texture on-the-fly in each frame
  - \+ texture is only used for visible area
- associate textures with quadtree nodes
  - \+ improved quality
  - \- many context switches required
- apply perspective reparametrization taking into account the current viewpoint (similar to perspective shadow maps)
  - \+ reduces aliasing artifacts

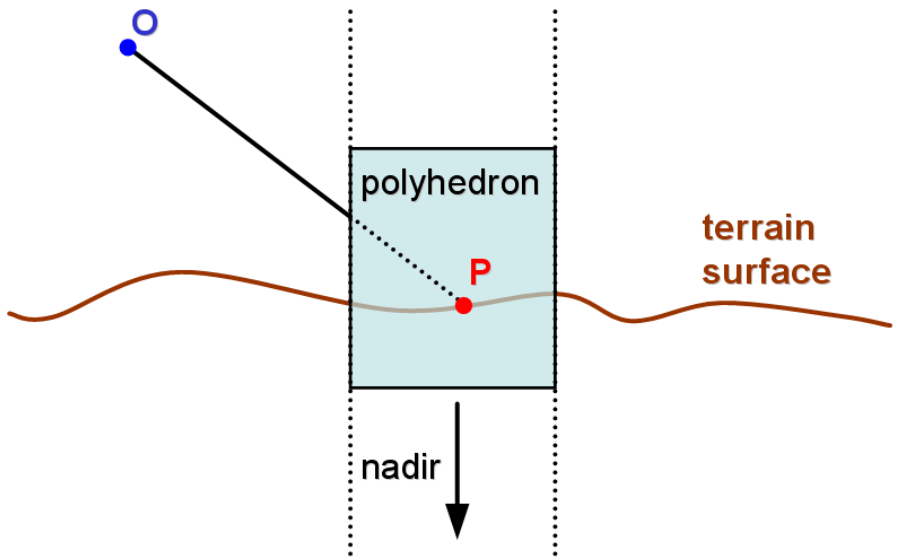
Problem Formulation

- rendering of vector data requires determination of projection onto terrain surface

- problem of determining the projection area can be interpreted as **point-in-polyhedra problem**

Point-In-Polyhedra Test

- choose point O outside all polyhedra
- for a point P in question count the intersections between the line segment OP and the polyhedra
- increment on enter, decrement on exit
- final count corresponds to number of polyhedra containing P



Point-In-Polyhedra Test

- notice the relation to shadow determination problem (shadow volumes) that can also be interpreted as a point-in-polyhedra-problem
- perform hardware accelerated point-in-polyhedra test with stencil buffer
  - each pixel is interpreted as a point P
  - count intersections by rasterizing front- and back-faces into stencil buffer

Point-In-Polyhedra Test

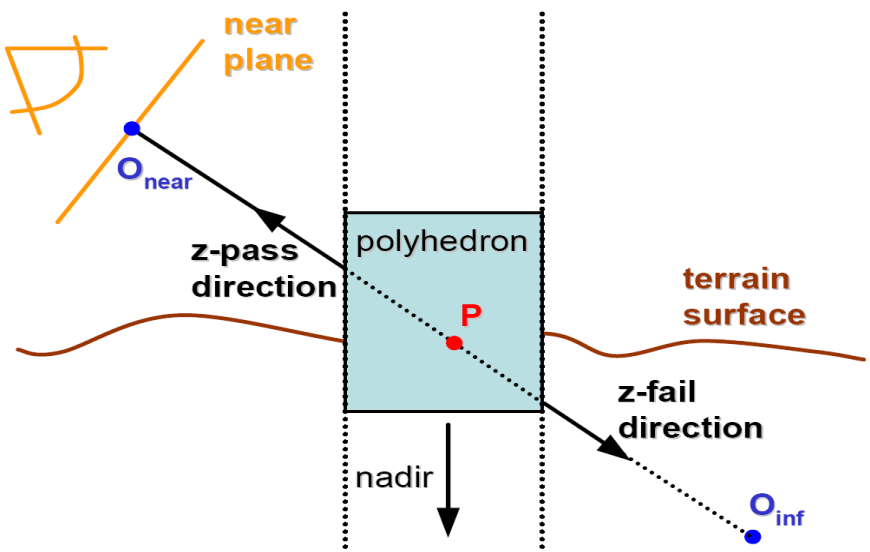
Two possible choices for point O:

O_{near} at the intersection of the ray and the near clipping plane

- count only fragments that pass the depth test
- increment/decrement for front/back-faces

O_{far} at infinity at the far end of the ray

- count only fragments that fail the depth test
- decrement/increment for front/back-faces



Point-In-Polyhedra Test

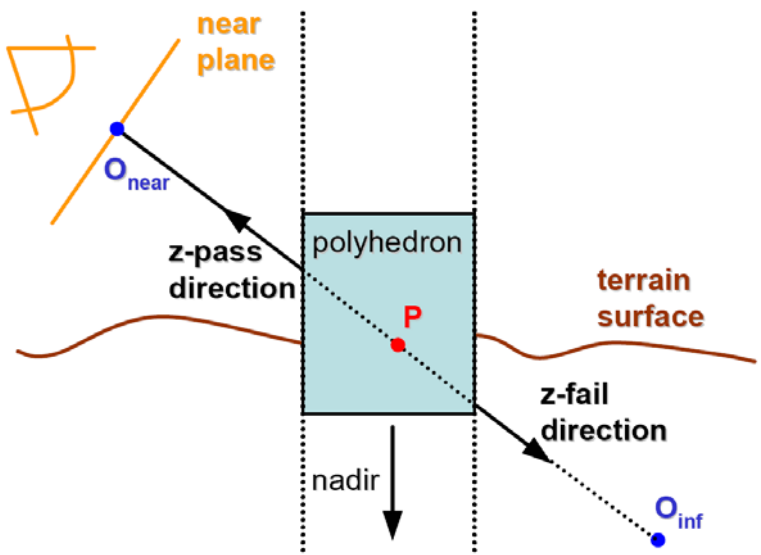
z-pass

- needs to render sides and top cap
- does not work correctly when near clipping plane intersects polyhedron

z-fail

- needs to render sides, top and bottom cap
- avoids far plane clipping by moving the far plane to infinity

if near clipping plane intersects view frustum
➔ use z-pass
else
➔ use z-fail



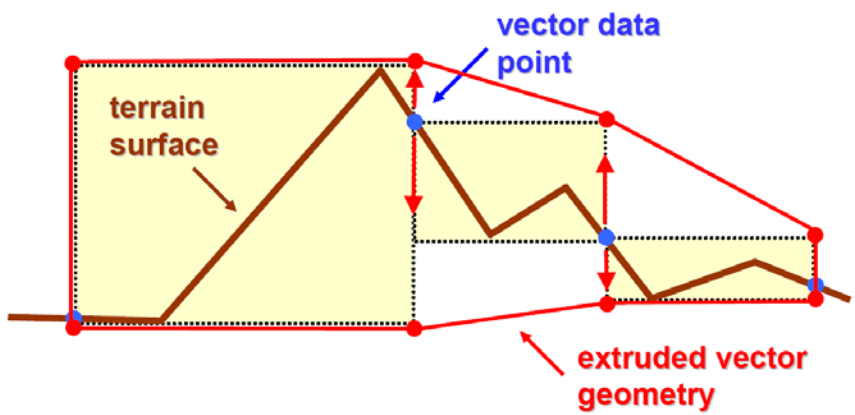
Our Method

Outline of the algorithm:

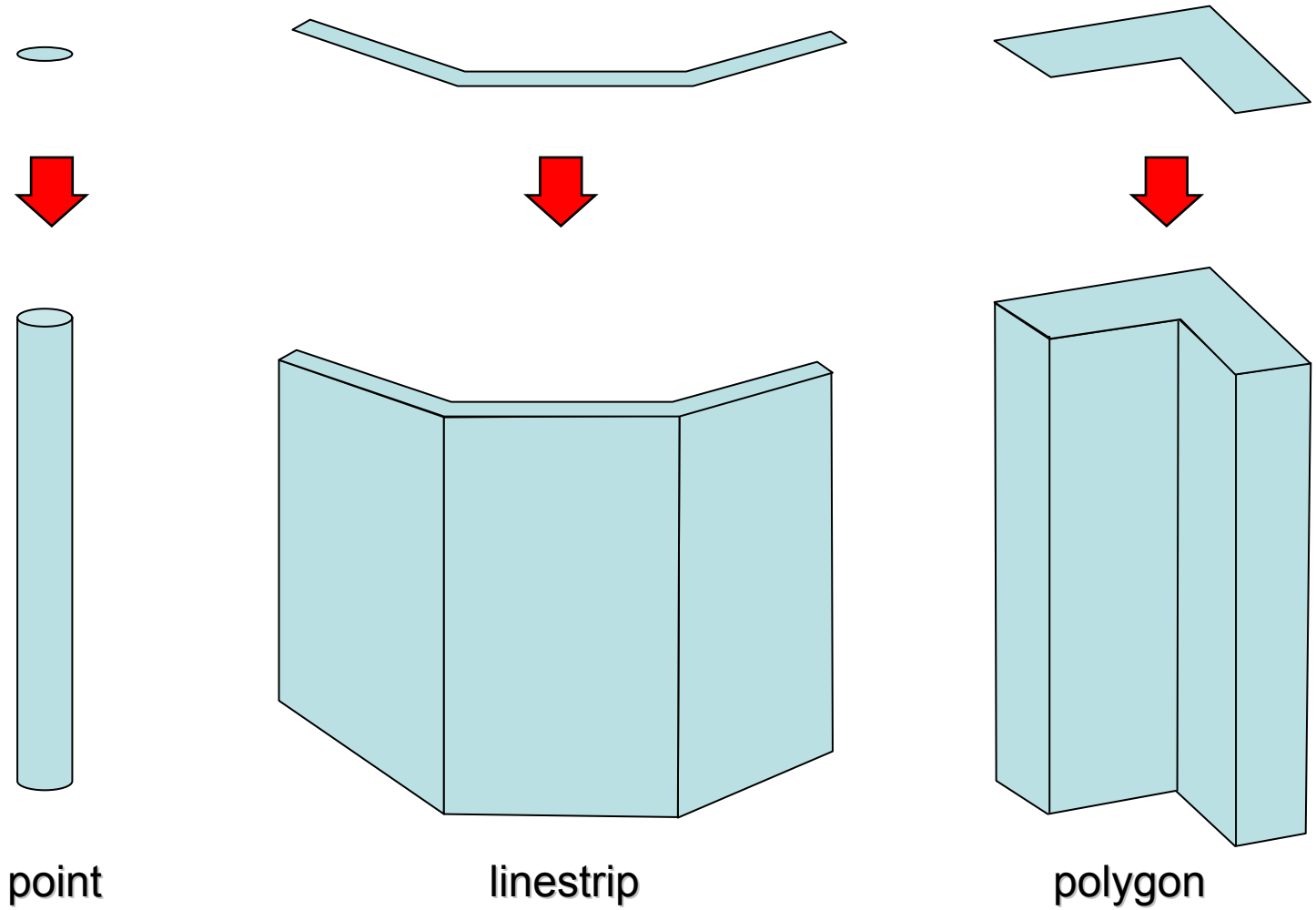
- vector data extrusion
  - create polyhedra from the vector data
- mask creation
  - render polyhedra into the stencil buffer
- apply mask to the scene
  - render geometry that covers at least the previously created mask

Vector data extrusion

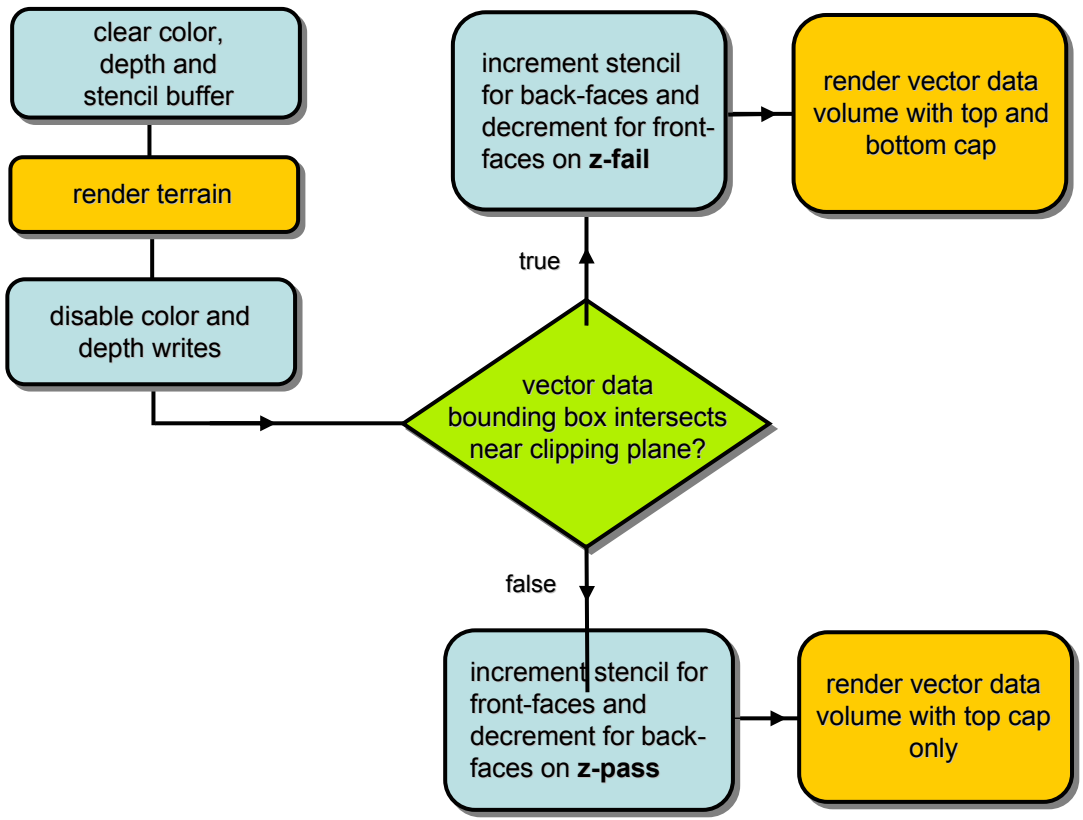
- duplicate each vertex in vector data
- move towards geocenter and in opposite direction
- reduce rasterization workload by reducing the size of the polyhedra
  - use minimum/maximum height of quadtree bounding boxes as lower/upper bound
- tesselate with consistent winding order and store in vertex buffer object



Vector data extrusion

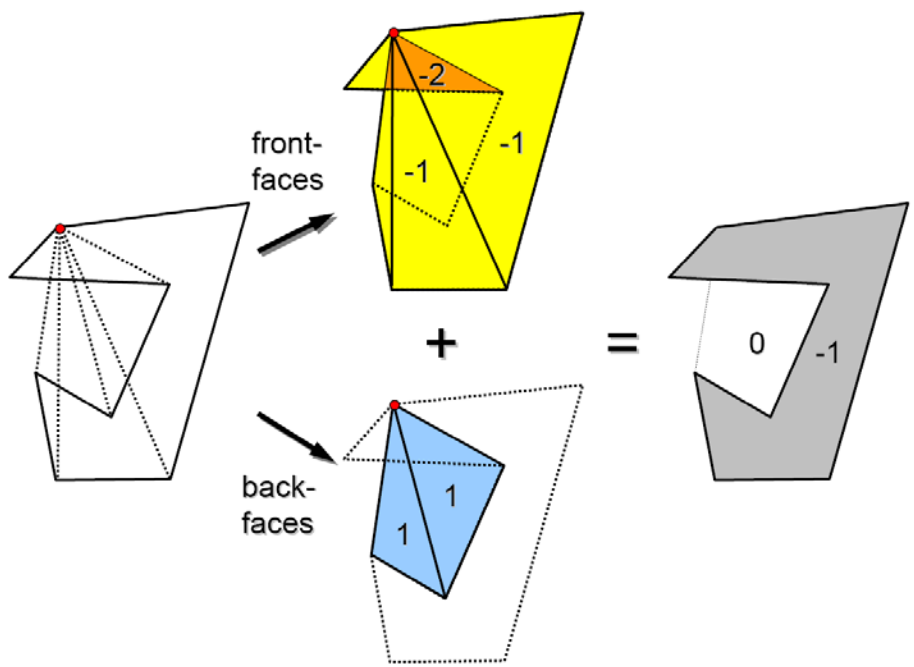


Mask generation

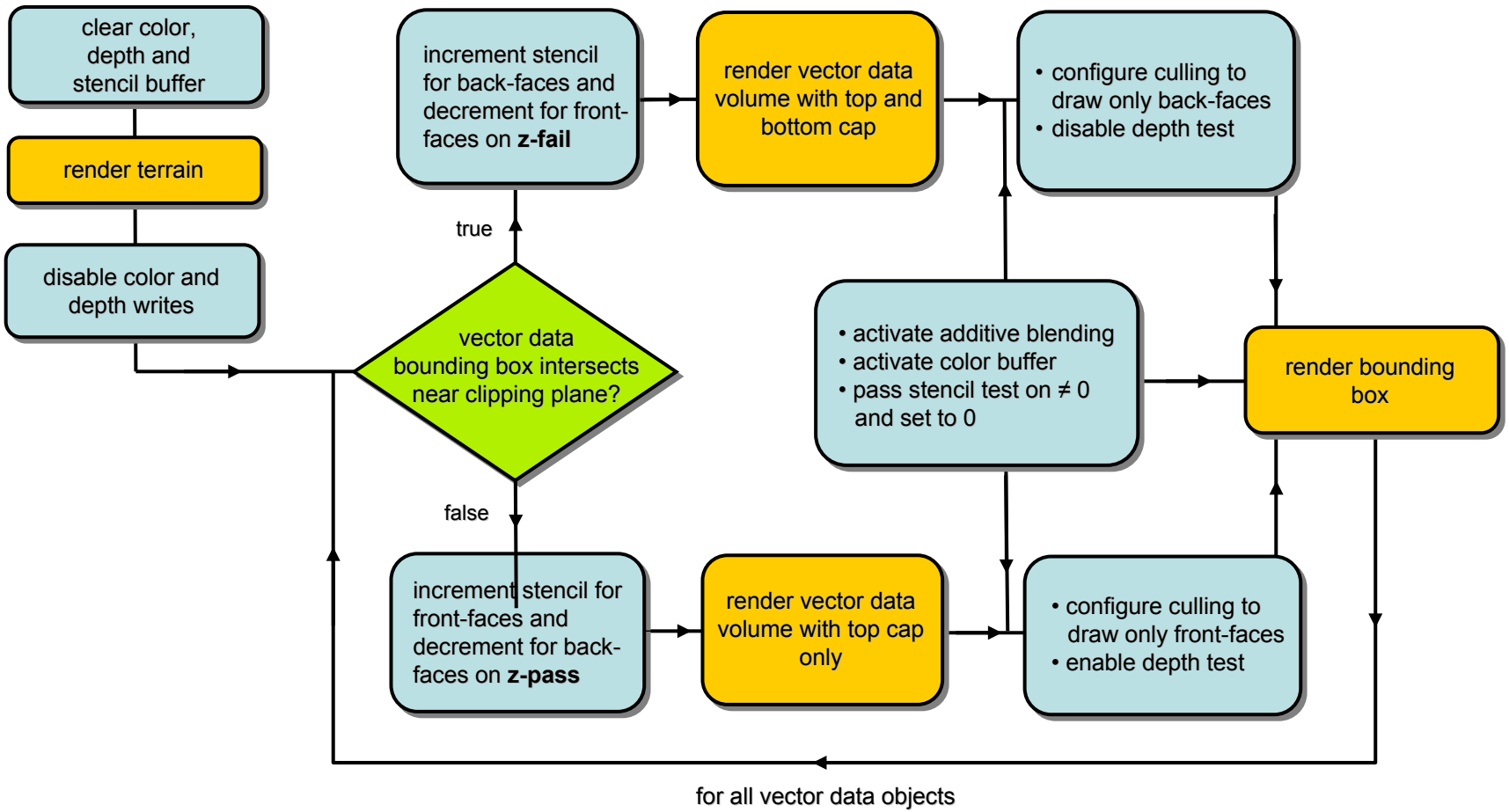


Mask generation

- simple triangle fan can be used to draw top and bottom caps
- triangle fan itself may be convex but produces the correct concave shape in the stencil buffer
- similar to computing signed area of a polygon by constructing a fan
- no need to triangulate in advance



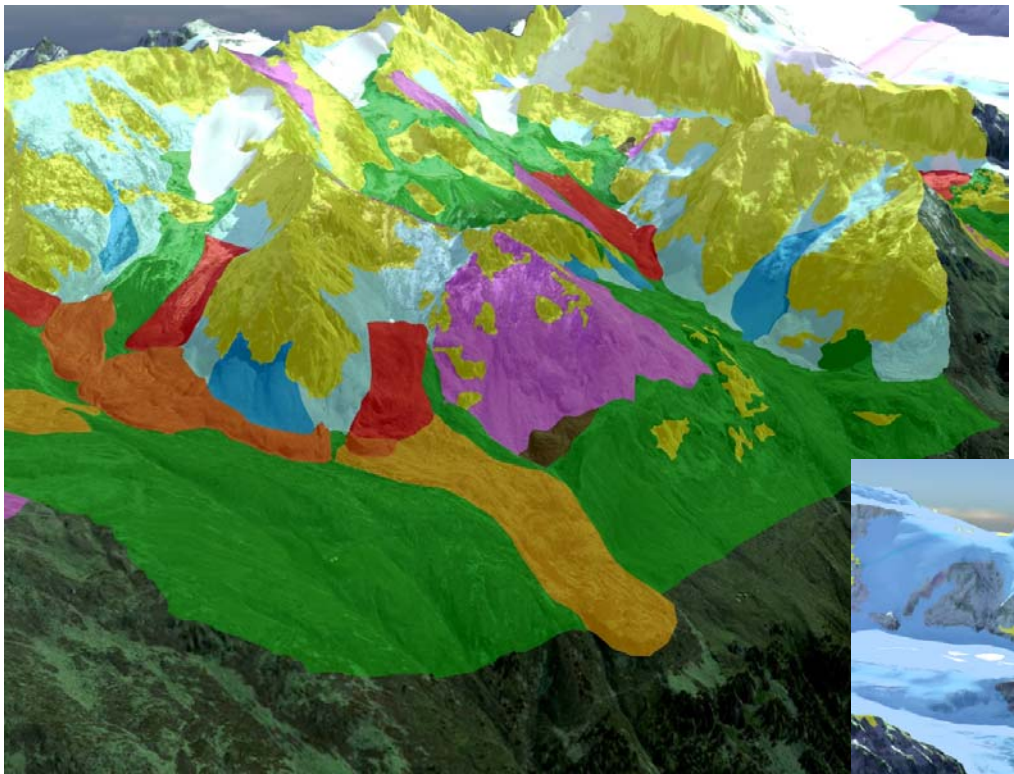
Mask application



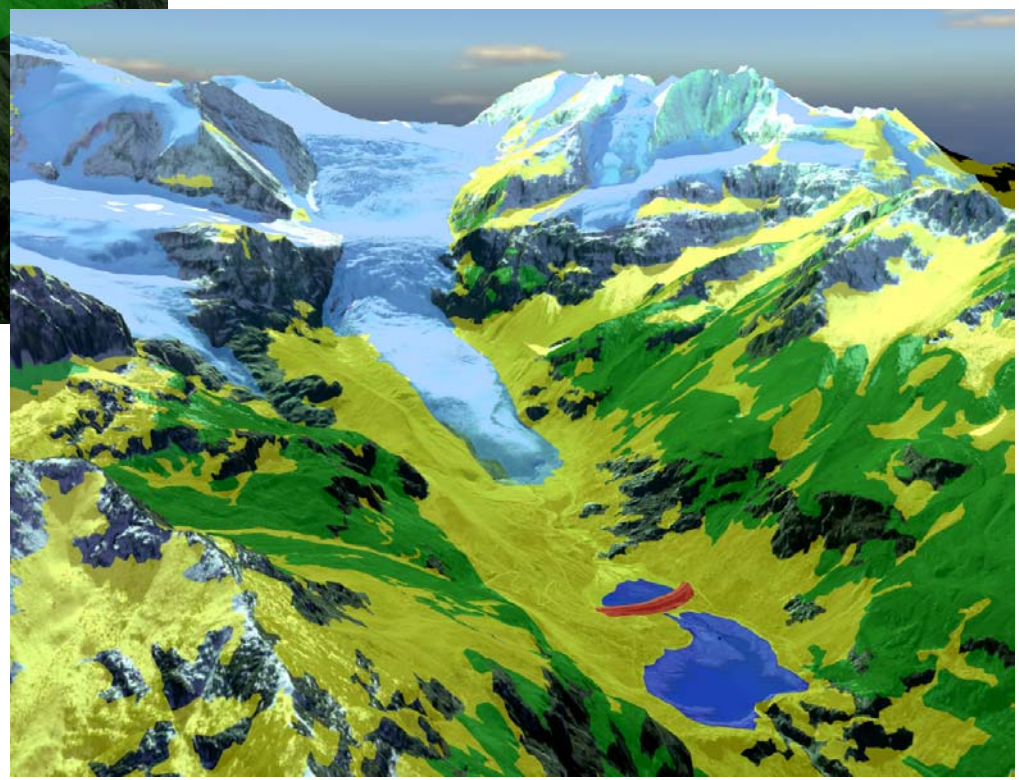
Results

Roads and trails in the Wettersteingebirge

Results



- Geomorhological mapping in Turtmann valley (Switzerland)
- ~100,000 vertices



- Glacier, lake and different soil types
- ~650,000 vertices

Conclusions

+ quality superior to texture-based methods and comparable to geometry based methods
+ independent of underlying terrain engine
  - allows easy integration in any terrain visualisation engine
+ independent of terrain complexity
  - can be used with upcoming ever-higher resolution data sets
- needs to render more primitives than texture based methods

Future Work

- LOD schemes
- distortions in steep slopes
- texturing and lighting

Thanks for your attention!

Any questions?