GDC | March 21-25, 2022
San Francisco, CA

Designing the Terrain System of Flight Simulator: Representing the Earth

Lionel Fuentes – Technical Director – Asobo Studio

#GDC22

ASOBO STUDIO

Microsoft Flight Simulator

Who are we?

ASOBO STUDIO

- Since 2002 in Bordeaux, France
- ~250 people
- 2019: A Plague Tale : Innocence
- 2020: Microsoft Flight Simulator

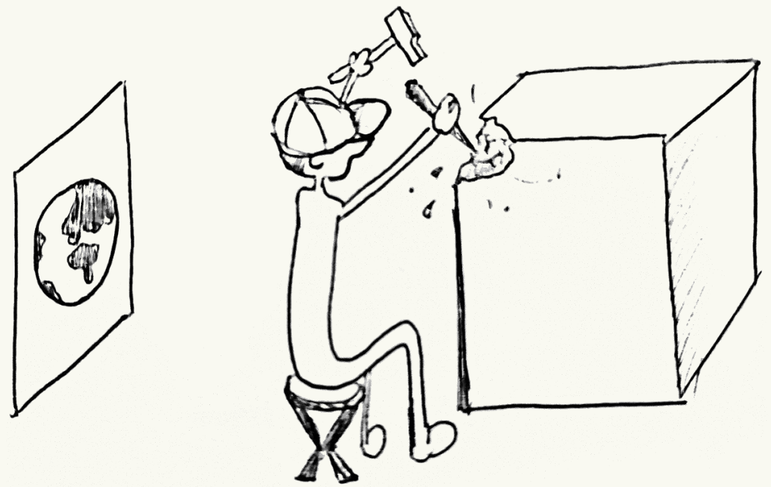

Microsoft
Flight Simulator

Agenda

- Introduction
- Terrain system architecture
- Flexibility
- Scalability

Problem statement

- Build the best possible 3D representation of the Earth
- Take off and land to and from every place on Earth
  - With a particular emphasis on airports and airfields
- Match the real world as much as possible



Available resources

- Bing Maps data
- OpenStreetMap data, other datasets
- Microsoft Azure
  - Compute
  - Storage
  - Streaming
- Original Flight Simulator X code and assets
  - Airports rendering code
- Artists

Data types

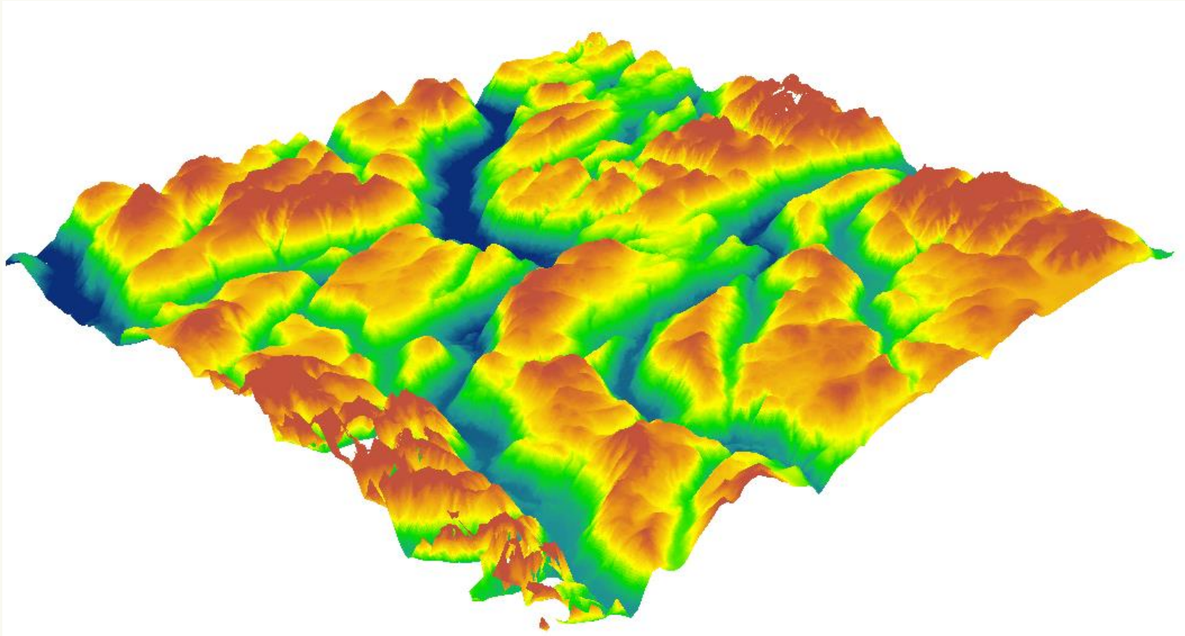
TIN:

- Triangulated Irregular Network
- Issues from photogrammetry

Data types

DEM:

- Digital Elevation Map
- Ground height estimate



Data types

Aerial images:

- Photos taken from satellites or aircraft

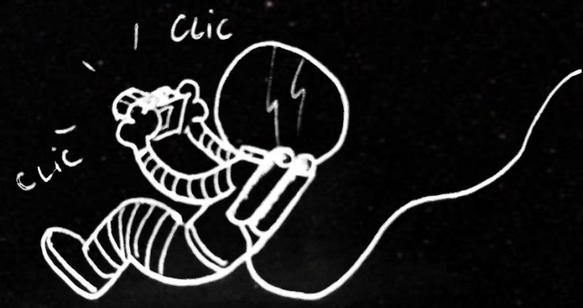
Data types

Vector data:

- Points, lines, polygons
- Used to represent:
  - Water bodies
  - Roads
  - Building footprints
  - Individual trees

Challenge #1: the world is big

- 510.1 millions km²
- 2+ million cities
- 1.5+ billion buildings
- 2+ petabytes of aerial images
- ~37,000 airports



Challenge #1: the world is big

- Limited manual edition
  - Semi-procedural systems
- Lots of storage needed
- Streaming!

Challenge #2: data quality

- Varying levels of quality
  - Aerials and DEM precision
  - Photogrammetry only available in a limited number of locations
- Data issues
  - Baked-in clouds and shadows
  - Missing data
  - Inconsistency
    - Photos taken at different times of day/seasons/years, using different cameras…
- Limited control over input data quality

Challenge #3: building a platform

- Support add-ons:
  - From professional add-on developers
  - From hobbyists/community
  - UGC ⇒ No control over artistic authoring
- All kinds of add-ons:
  - Airports & airfields
  - Points of Interest (PoI)
  - DEM
  - Traffic enhancement, trees tweaks, boats…

Challenge #4: the world is evolving

- Regular Bing Maps updates
  - Should not break existing/published add-ons!
- Live game
  - “World Updates”
- Seasons, tides, climate change, cities evolution, tectonic plates…

Problem statement: v2

A) Make a terrain system that is **flexible**:

- Accepts different sources of data
- Can be easily modified outside of the main dev team
- Tolerates data updates
- Tolerates wrong, bad quality and/or inconsistent data

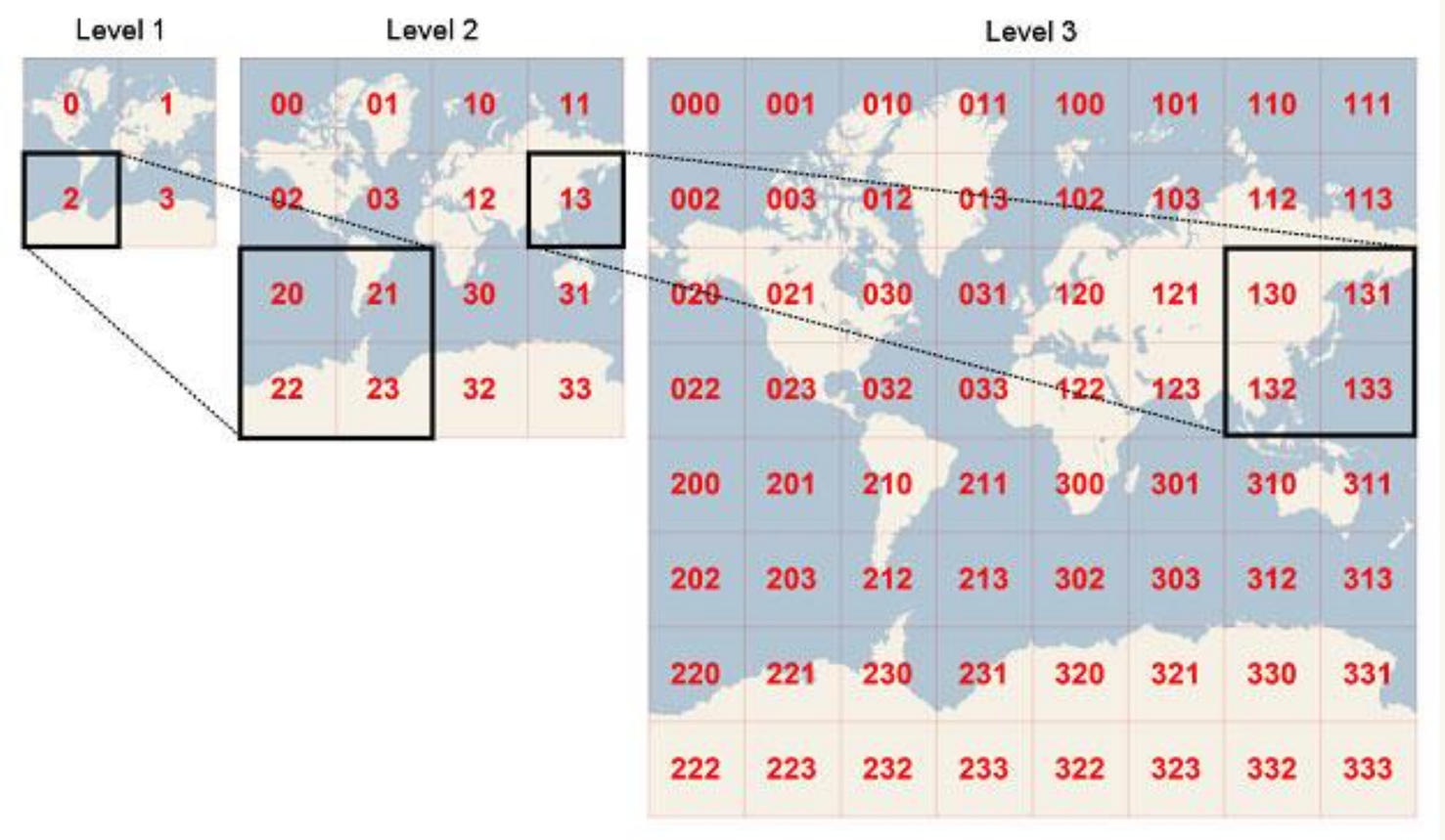
B) Make a terrain system that **scales**:

- Handles lots of data
- Displayed from any altitude
- Scale with hardware: performance and memory
- Scale with available Internet bandwidth


Terrain system architecture

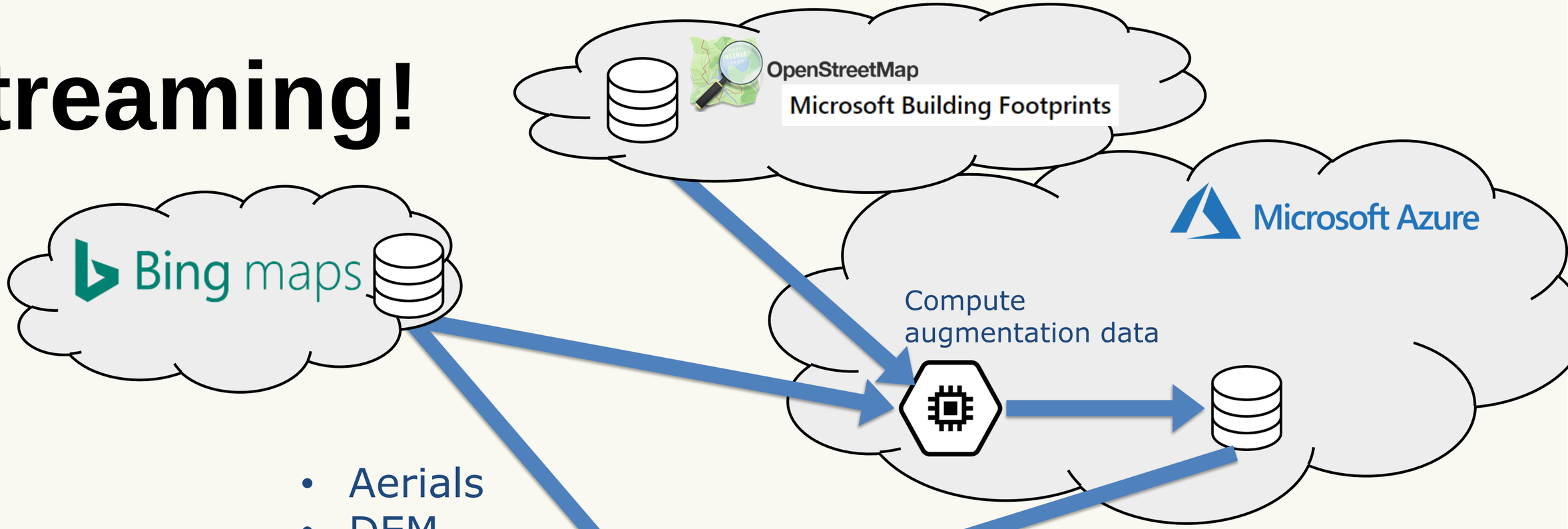
Data layout

- Quadtree
- Mercator projection
  - Stretches at the poles
- Not ideal but the vast majority of available data uses it
- LoD = number of subdivision steps



Bing Maps tiles system [1]

Streaming!



- Aerials
- DEM
- TIN

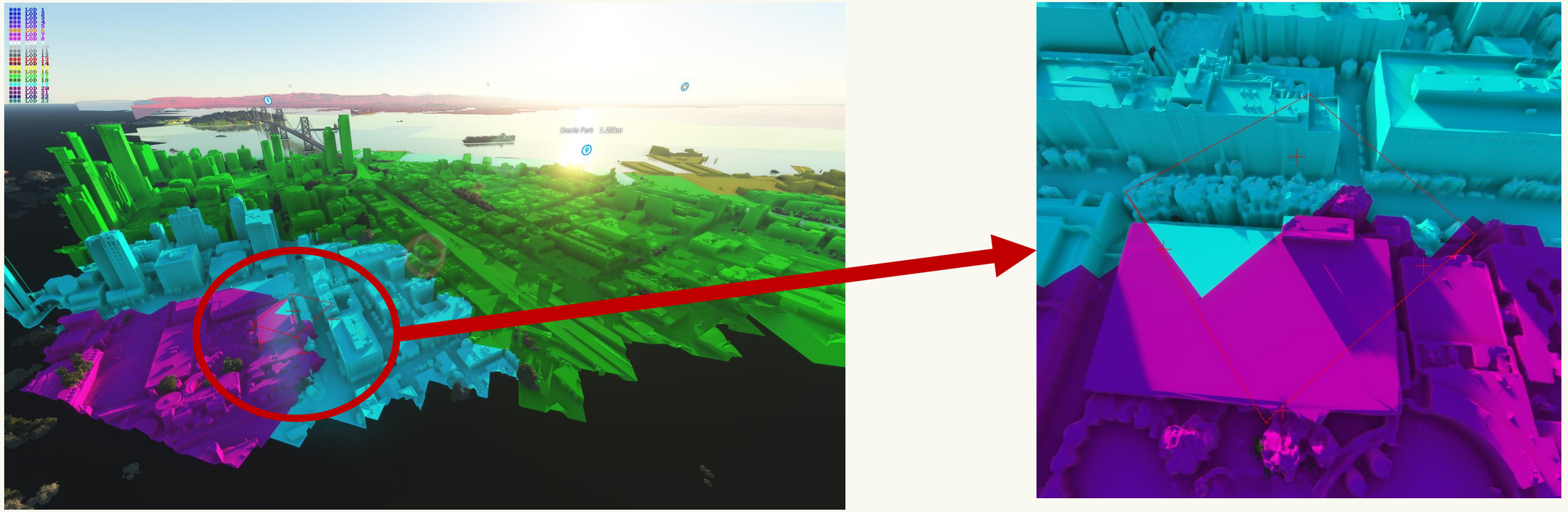
- Trees masks
- Roads
- Buildings footprints
- Water data
- …

Requesting and drawing tiles

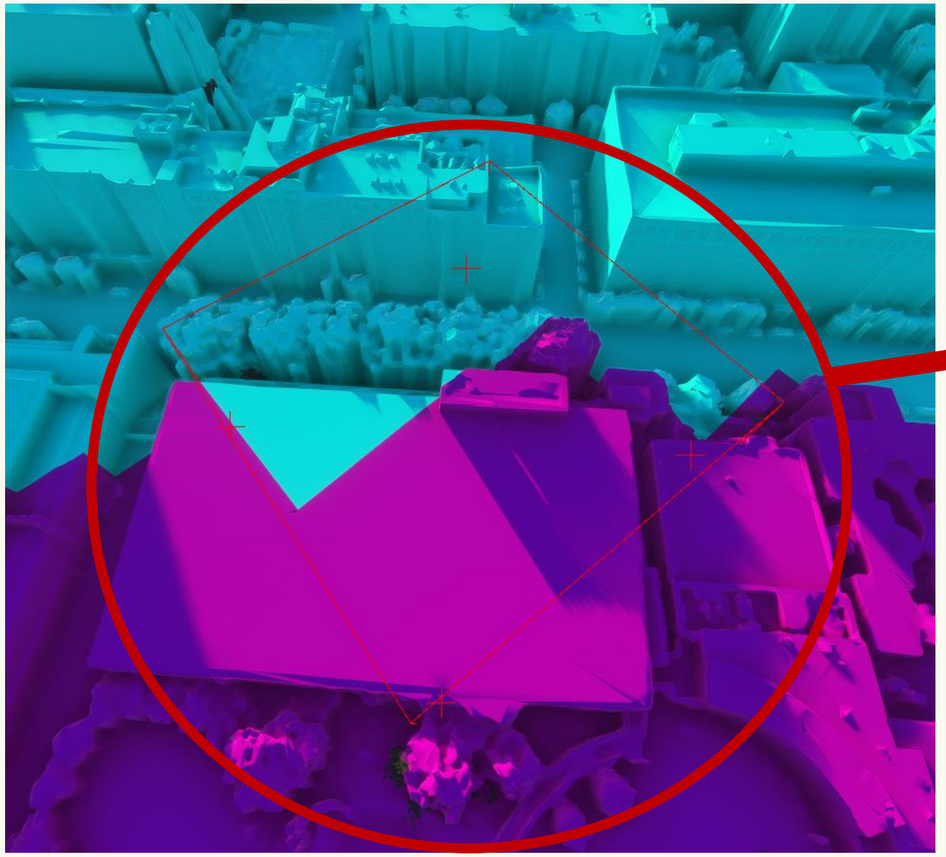
- `RecurseRequest()`
  - Recursively request tiles in view frustum
    - If not precise enough, increase requested LoD
  - Requested LoD depends on:
    - Estimated tile screen size
    - Estimated available network bandwidth
  - Maintain tiles around the camera in memory
- `RecurseDraw()`
  - Draw tiles using the best currently available data
    - Tiles can use data from a parent (e.g. aerial texture)

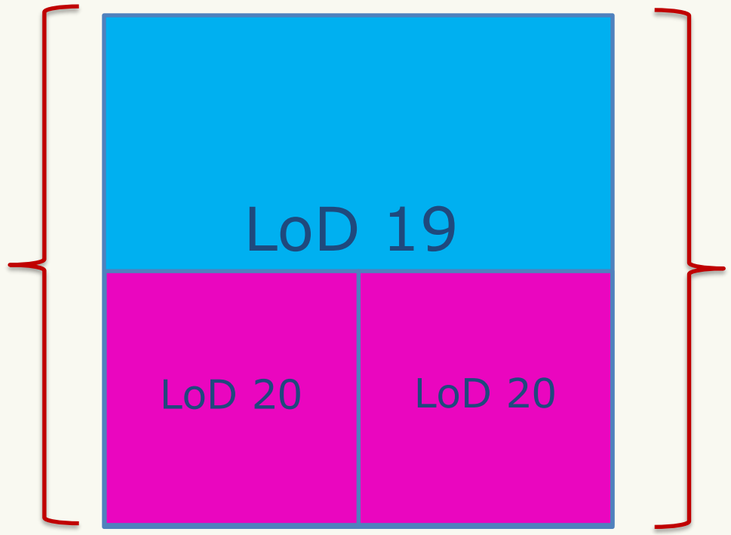
Cutting tiles

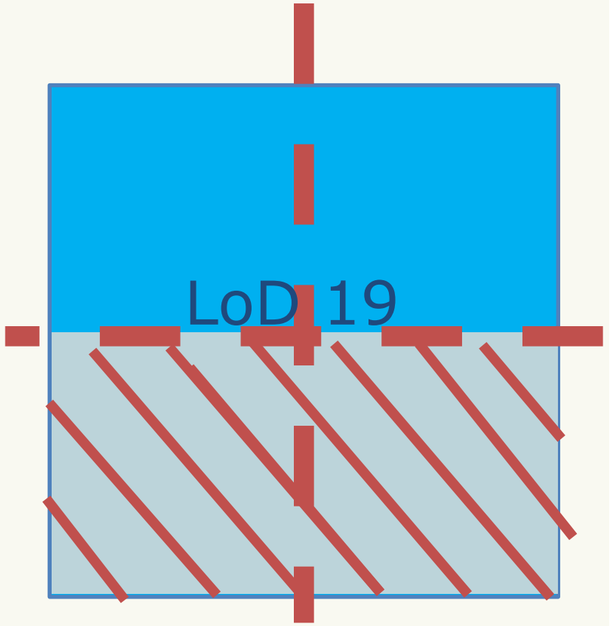
- No guarantee for when a tile becomes available ⇒ need to be able to do arbitrary cuts



Cutting tiles





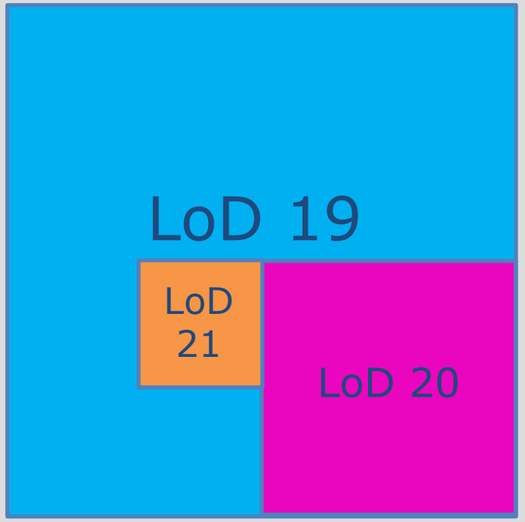


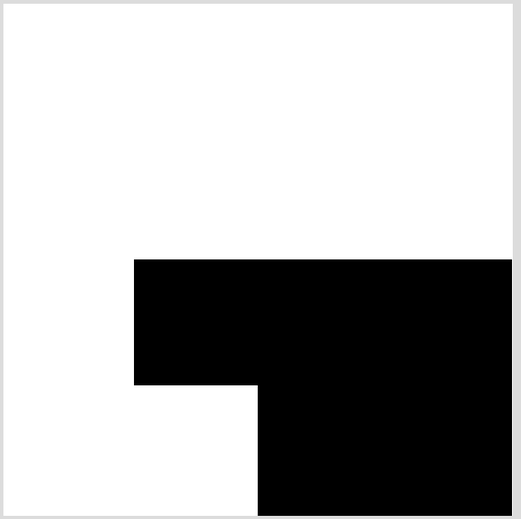
Cutting tiles

- Up to 2x2 granularity:
  - DEM tiles:
    - Choose appropriate pre-computed index buffer among global list of 15 index buffers
  - TIN tiles:
    - Generate additional vertices and index buffers
    - Choose appropriate pre-computed index buffer
- General case:
  - CPU-generated “tile mask” textures
  - Pixel shader:

```
if(mask is black)
        discard;
```

  - Works but expensive

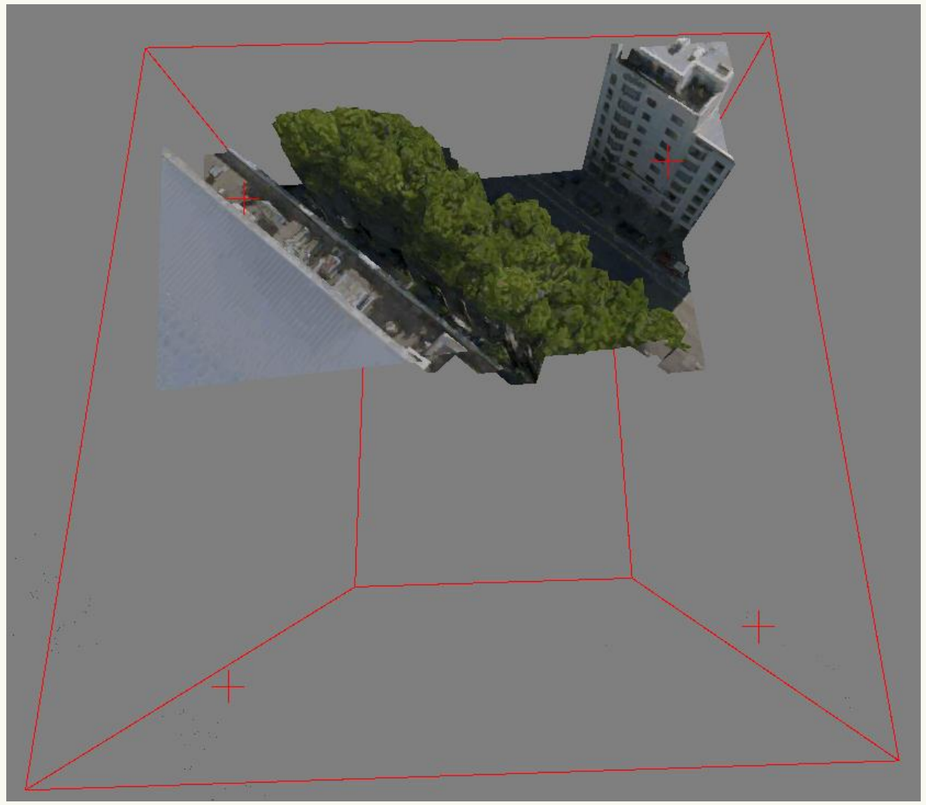


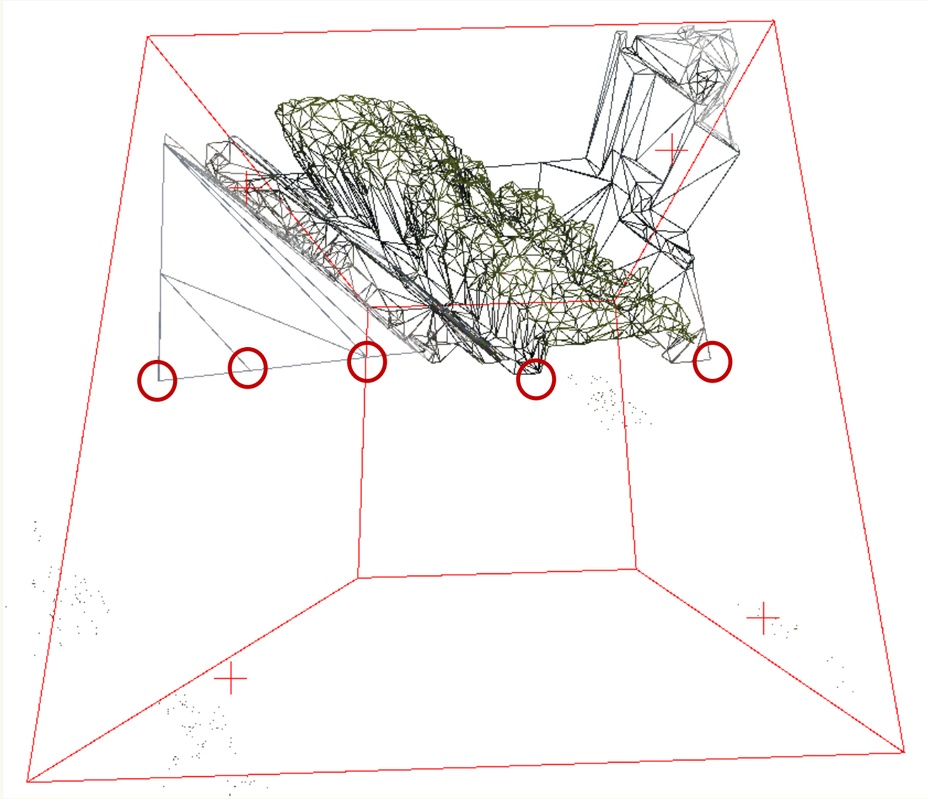


4x4 tile mask texture example

Cutting tiles

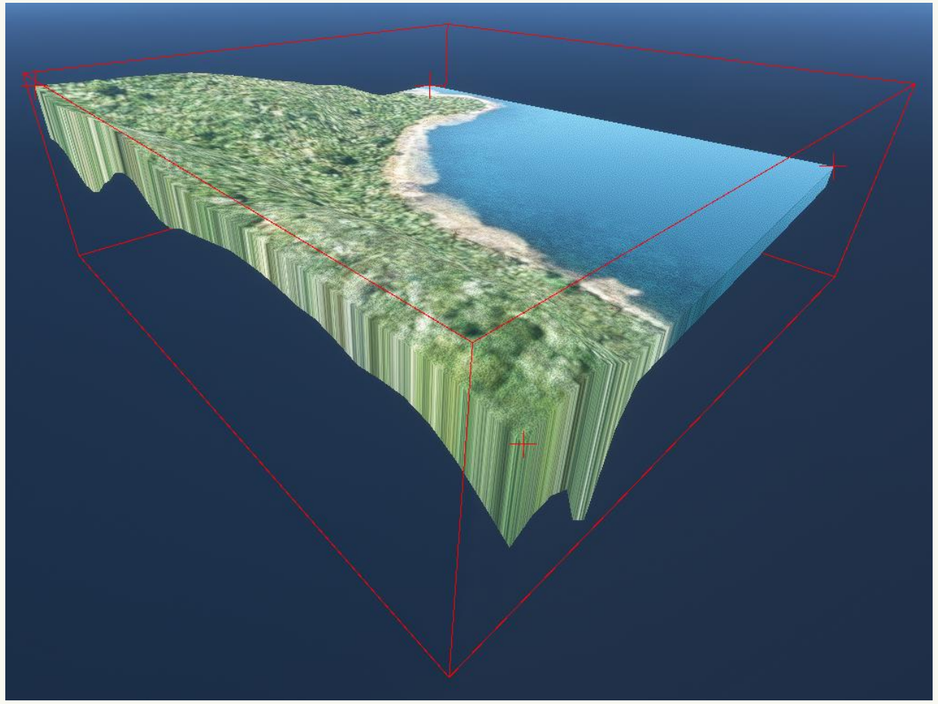
- TIN tile example





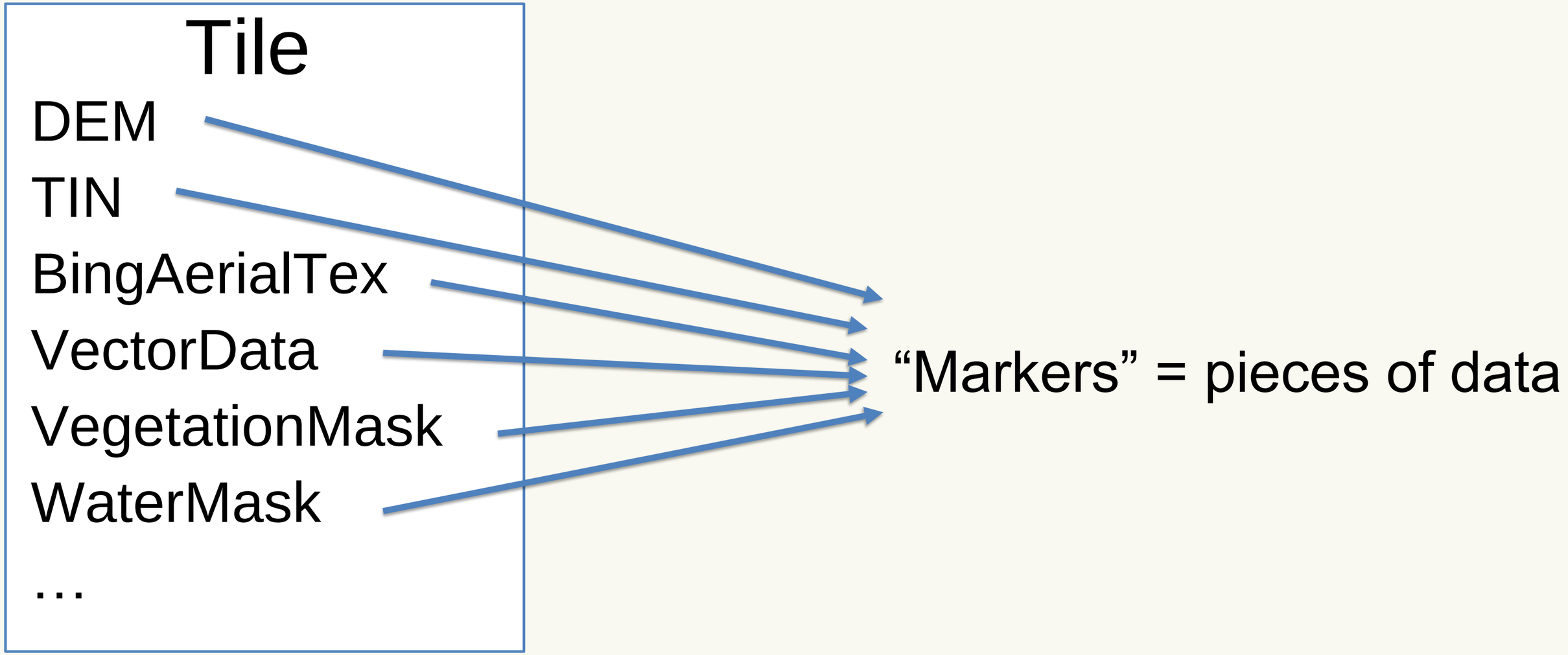
Tile skirts

- Used to hide cracks
- Doesn’t always work with tiles cutting
  - 2x2 case: generate skirts for sub-tiles
  - General case: pixel shader discard
    ⇒ no skirts!
- Skirt size depends on slope to reduce fill-rate




A) Make a terrain system that is flexible

Anatomy of a tile



Anatomy of a marker

- Each marker is a state machine:
  - Marker state:

    `REQUESTED`, `PROCESSING`, `AVAILABLE`, `FAILED`
  - Marker internal state (while `PROCESSING`): e.g:
    - Possible values depend on marker type (DEM, WaterMask, etc)
    - Example: `INIT`, `DRAW_TO_GPU`, `COPY_TO_CPU`, `MERGING`, `DONE`
- At the end of a step, the marker decides on its next step:
  - New CPU task (async or main thread)
  - Issue GPU draw calls
  - Issue a new HTTPS request
  - Set state to `AVAILABLE` or `FAILED`

Markers dependencies

- Markers can depend on other markers
- Markers request their dependencies
- Reloading system needed for:
  - In-game edition of the terrain
  - Reacting to new incoming data (airports)

Markers reloading

Steps:

Markers reloading

Steps:

1) Mark a marker as dirty

Markers reloading

Steps:

1) Mark a marker as dirty
2) Generate reloading marker (new version)

Markers reloading

Steps:

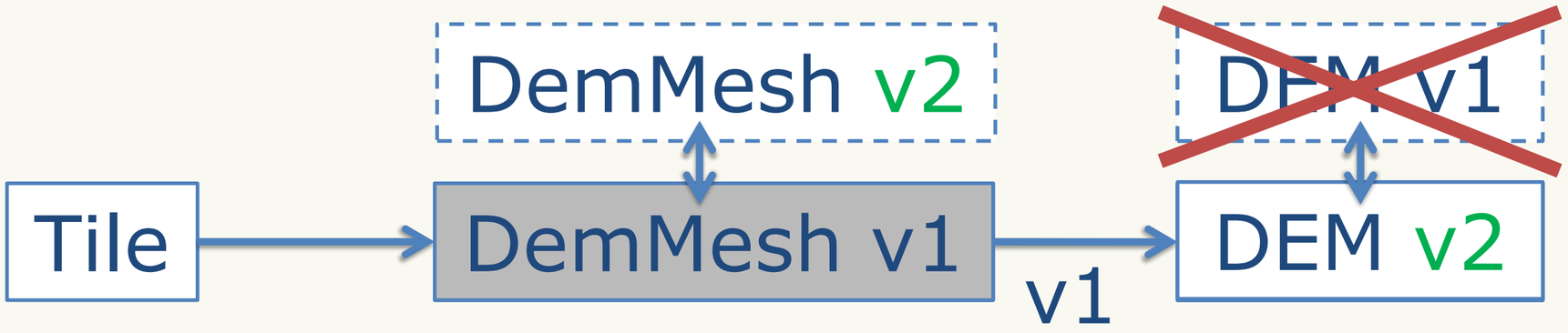
1) Mark a marker as dirty
2) Generate reloading marker (new version)
3) Process reloading marker until it is `AVAILABLE`

Markers reloading

Steps:

1) Mark a marker as dirty
2) Generate reloading marker (new version)
3) Process reloading marker until it is `AVAILABLE`
4) Swap its data and data version with the real marker and delete the reloading marker

Markers reloading

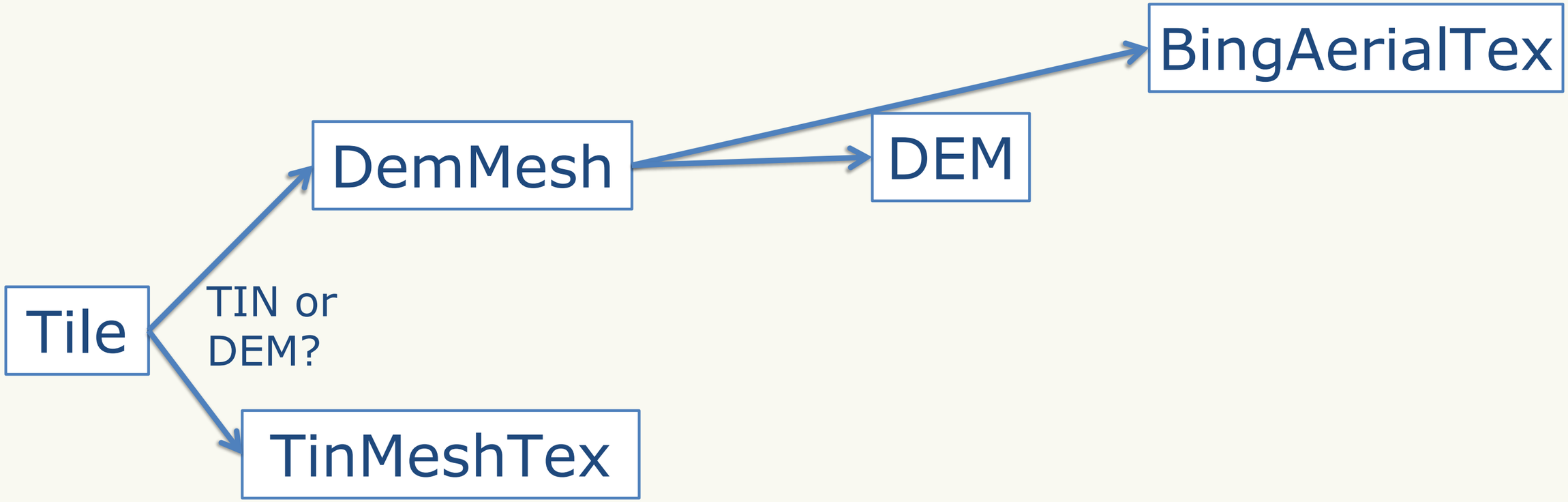


Steps:

1) Mark a marker as dirty
2) Generate reloading marker (new version)
3) Process reloading marker until it is `AVAILABLE`
4) Swap its data and data version with the real marker and delete the reloading marker
5) Trigger reloading for markers who expected a different data version in their dependencies

Markers in practice

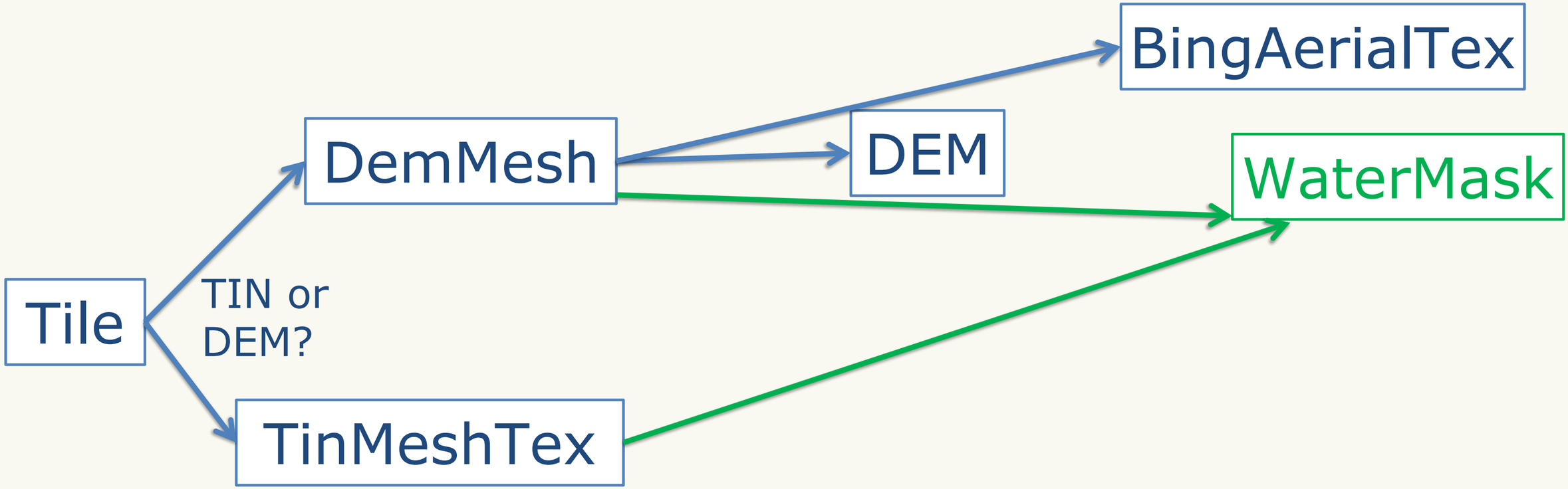
Markers in practice



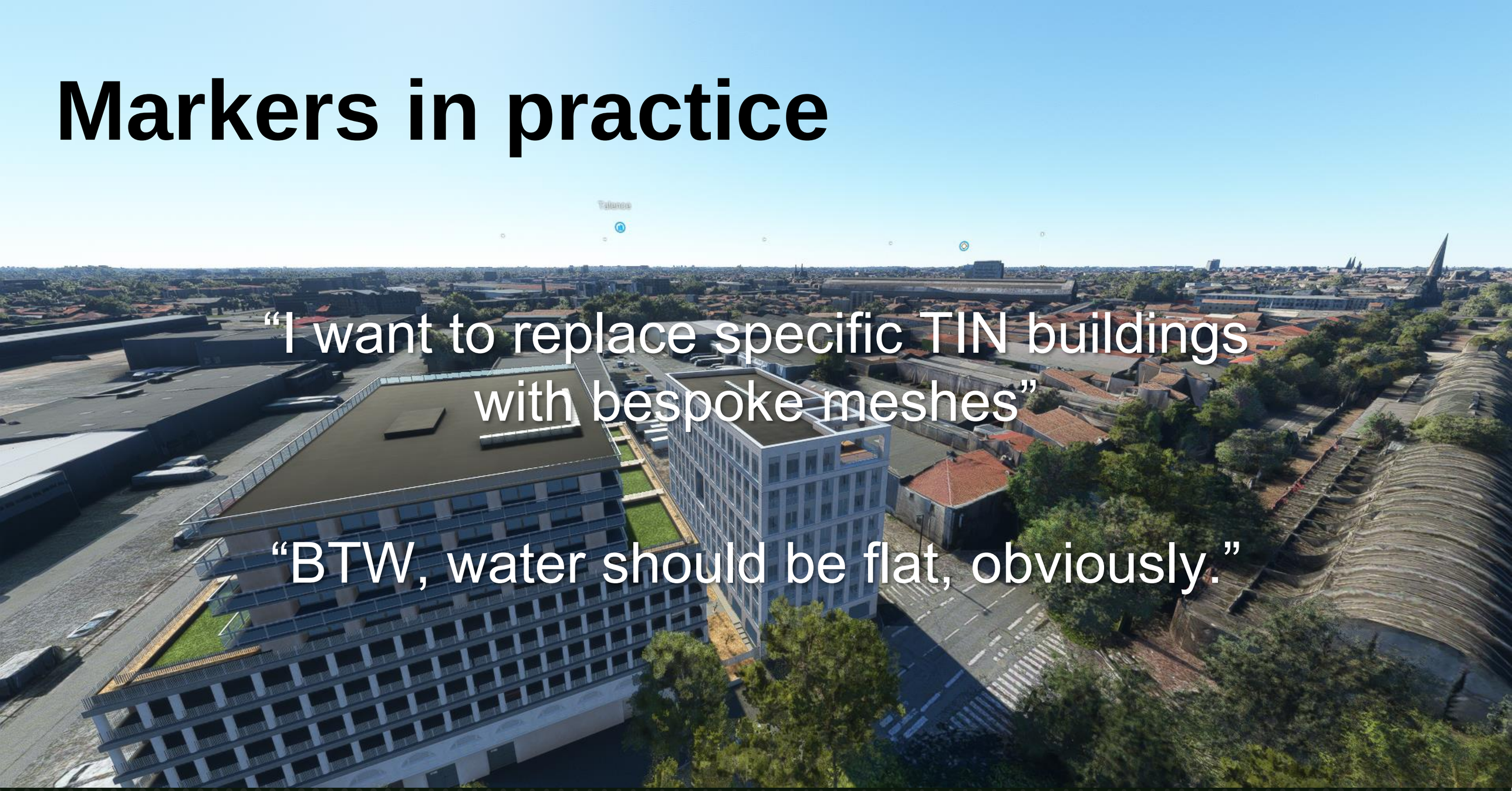
Markers in practice

"Terrain should feature a water effect in appropriate areas."

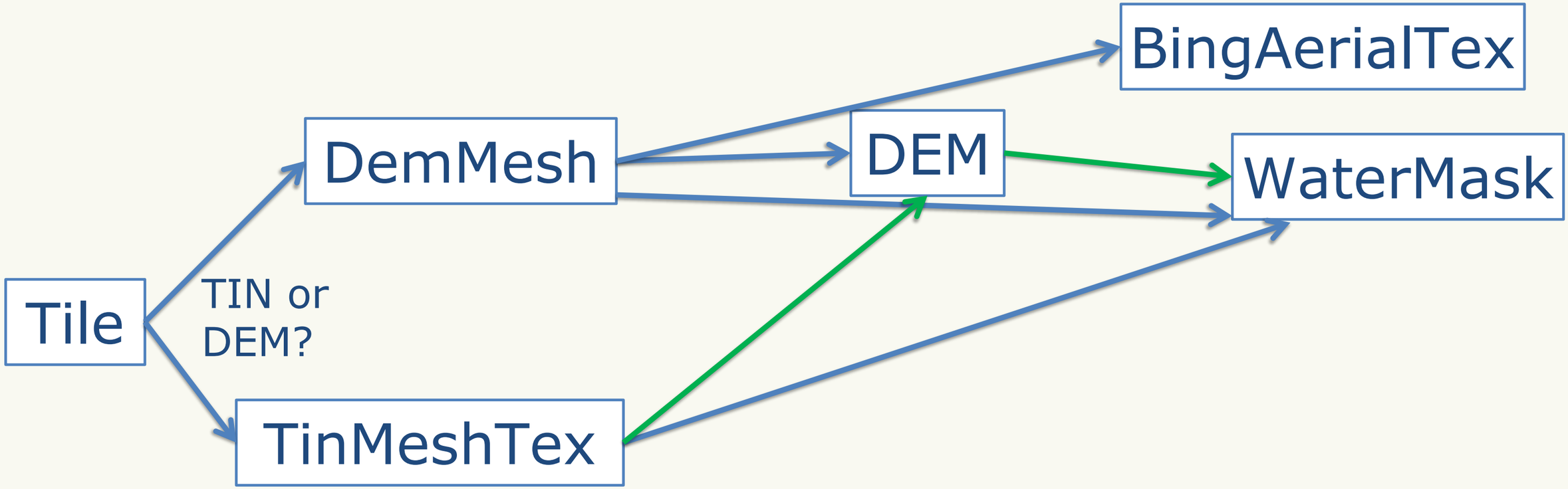
Markers in practice



Markers in practice

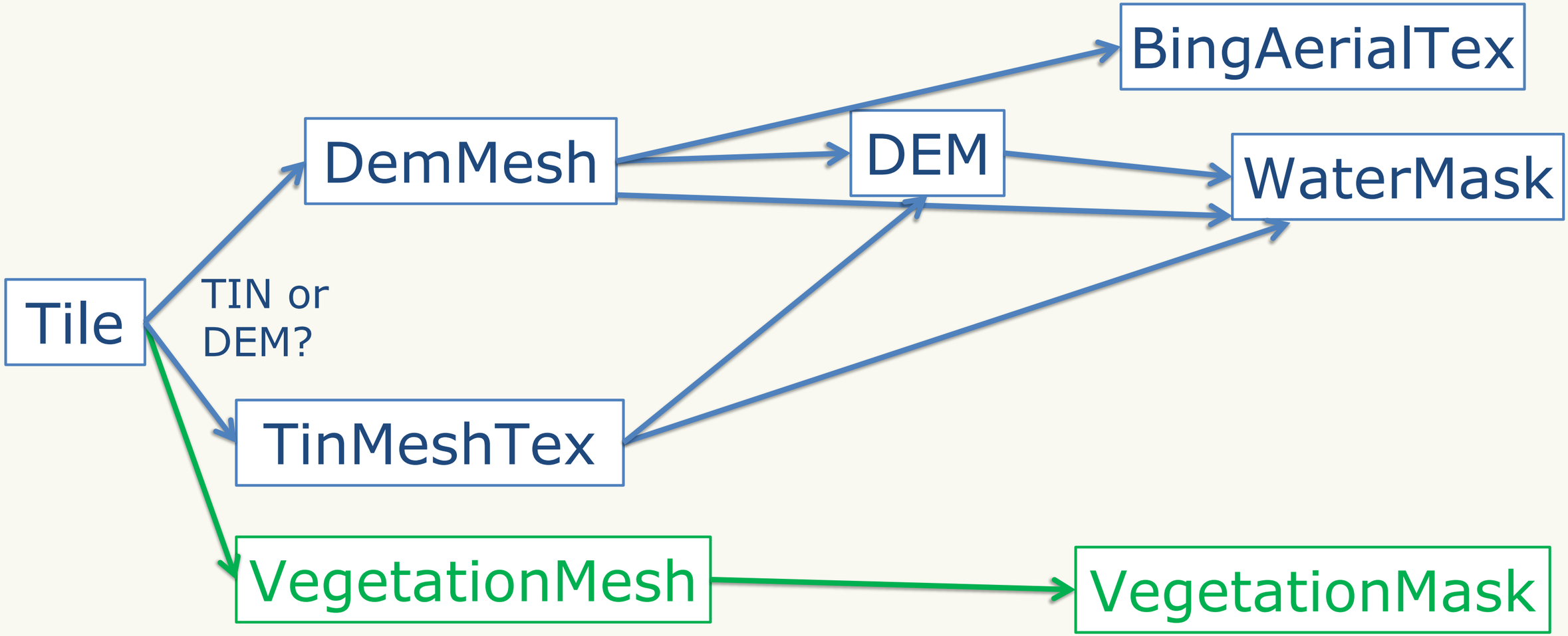


Markers in practice



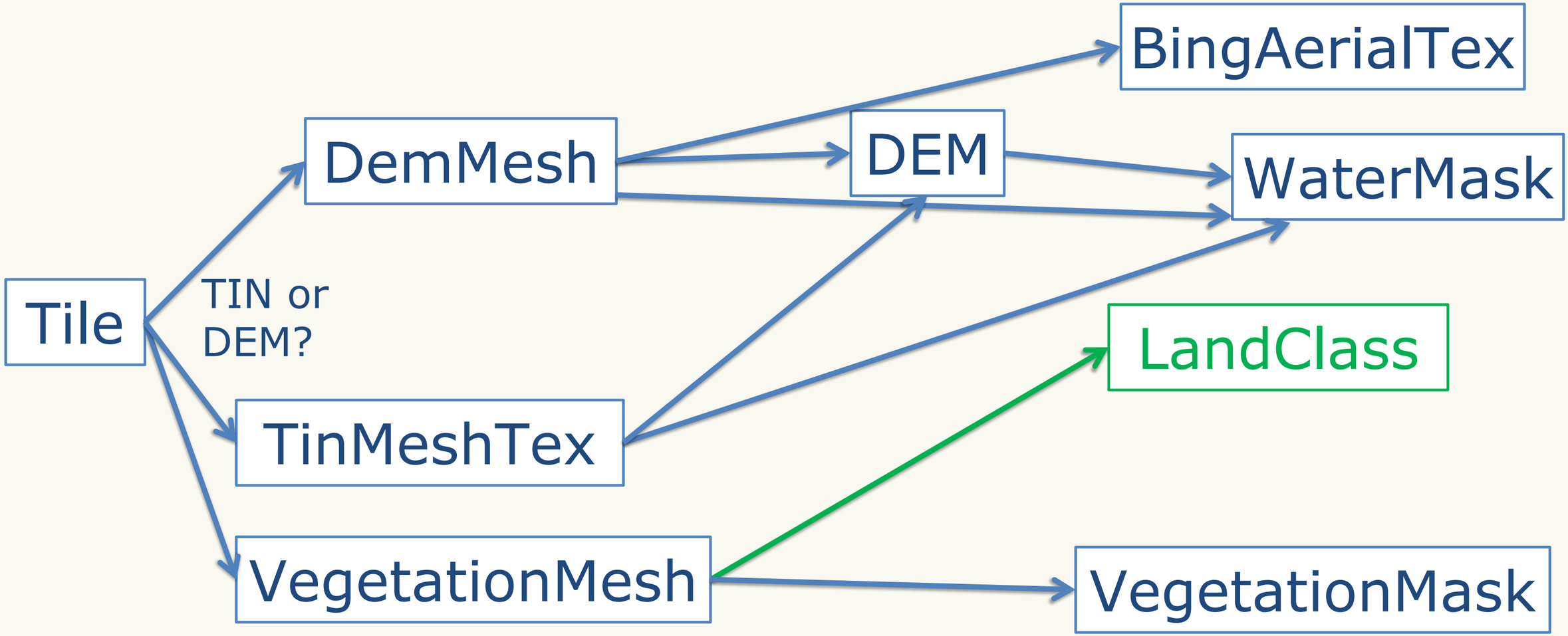
Markers in practice

Markers in practice



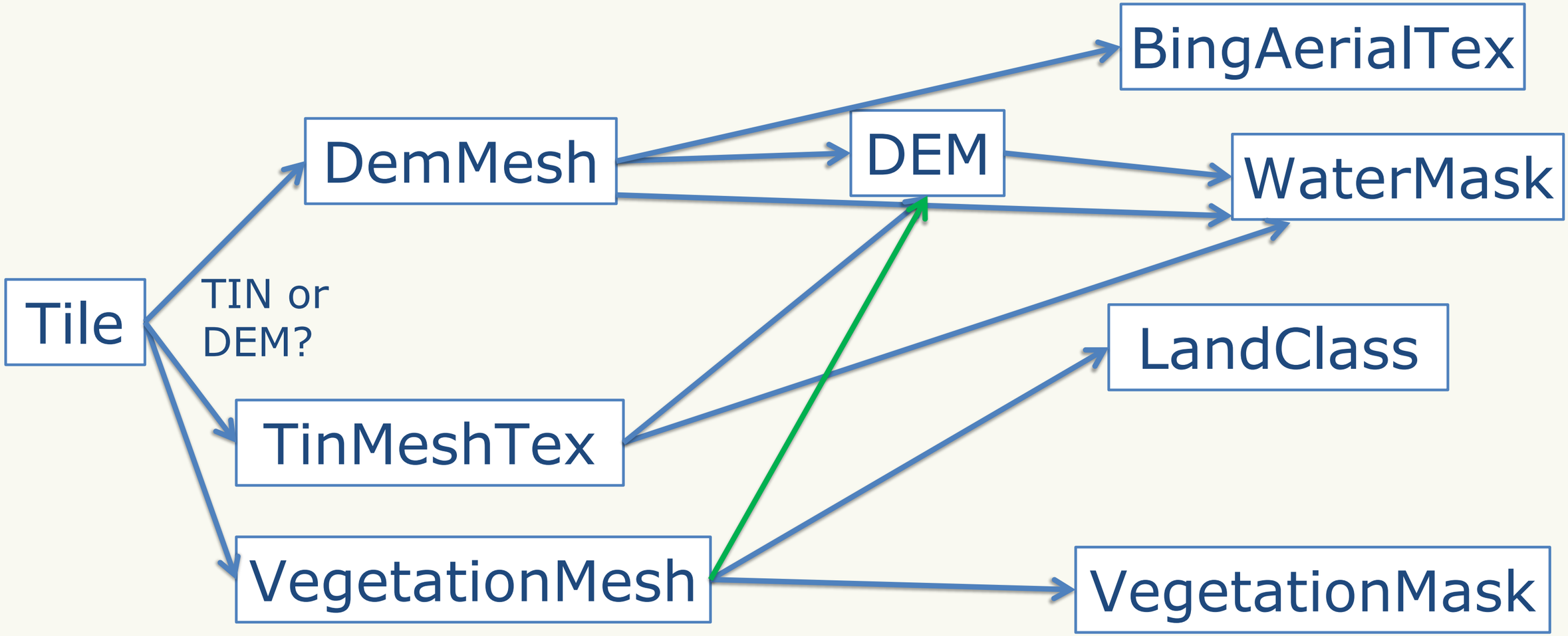
Markers in practice

Markers in practice



Markers in practice

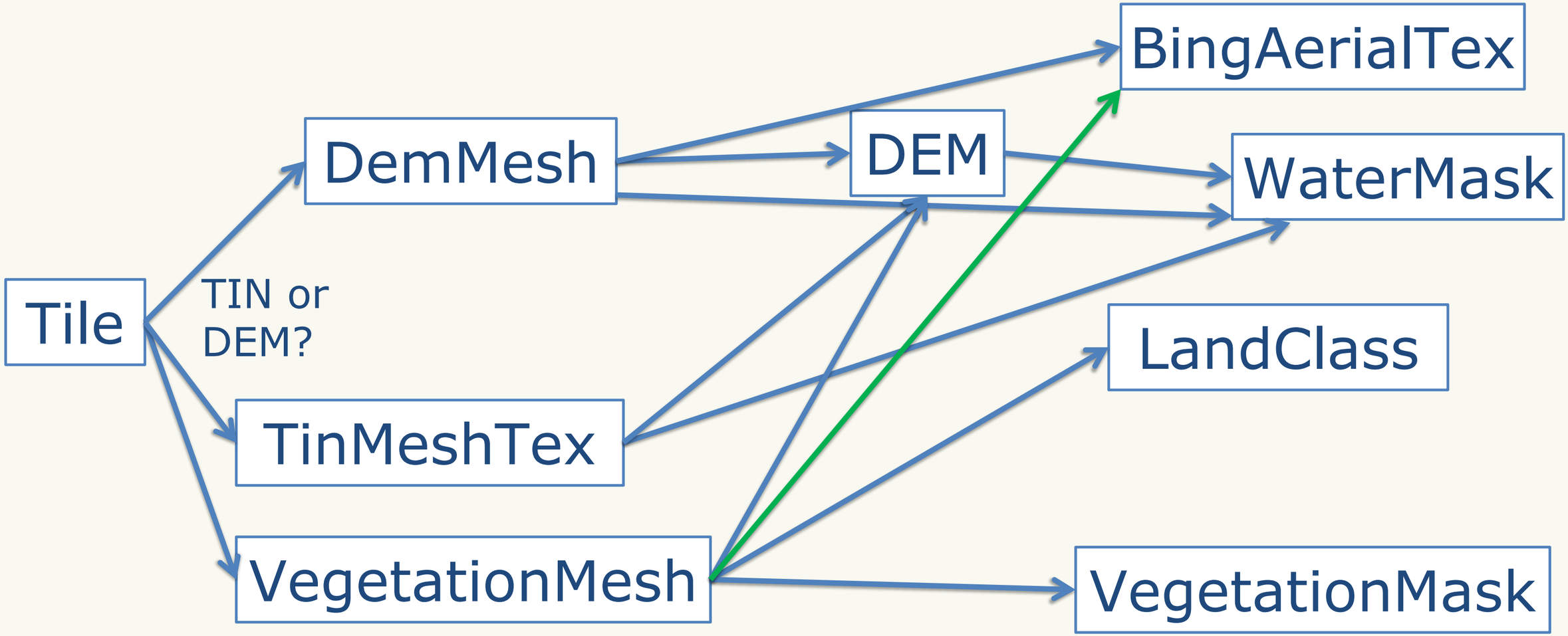
Markers in practice



Markers in practice

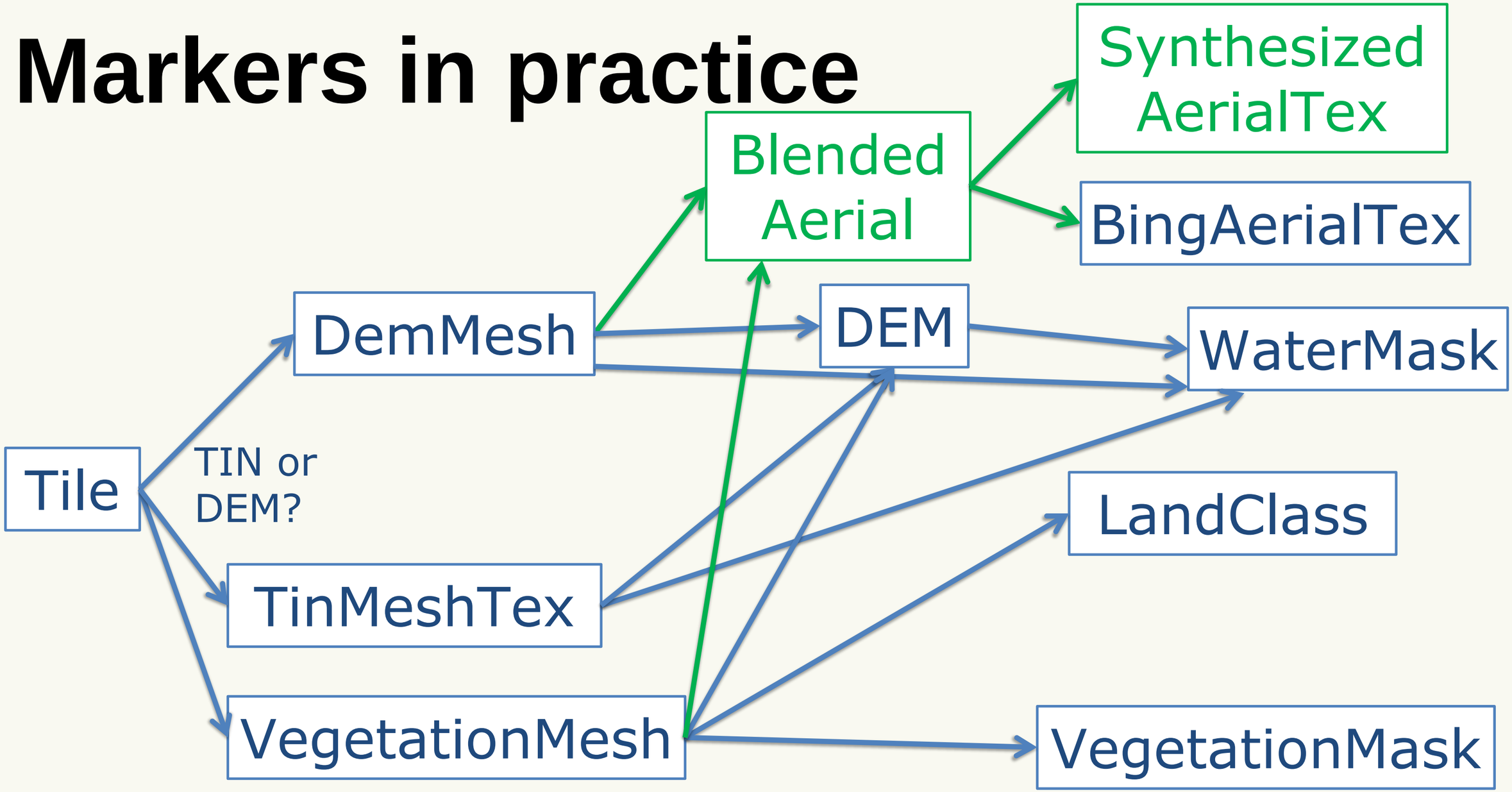
"Trees should be the same color as the aerial images"

Markers in practice



Markers in practice

Markers in practice

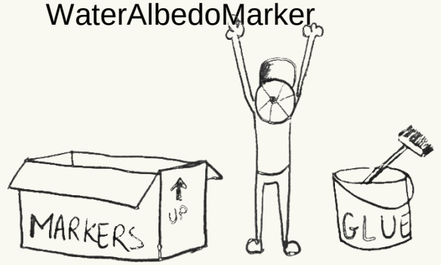


Other requirements

- “Draw airports on top of the terrain”
- “Grass should be colored according to the aerial”
  - “unless the ground color has been edited”
- “Synthesized aerials should be affected by altitude and slope”
- “Synthesized aerials should reflect the presence of trees”
- “There are no trees in water!”
- …

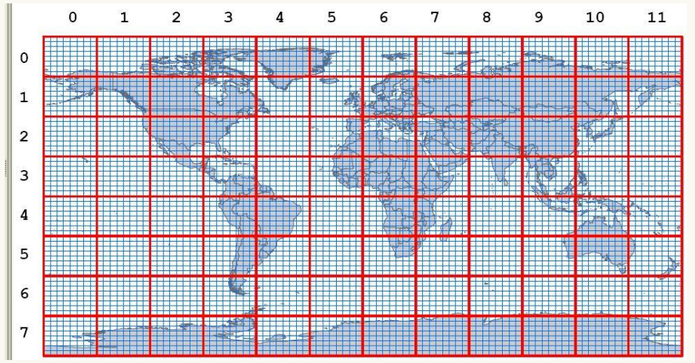
Actual list of markers (for reference)

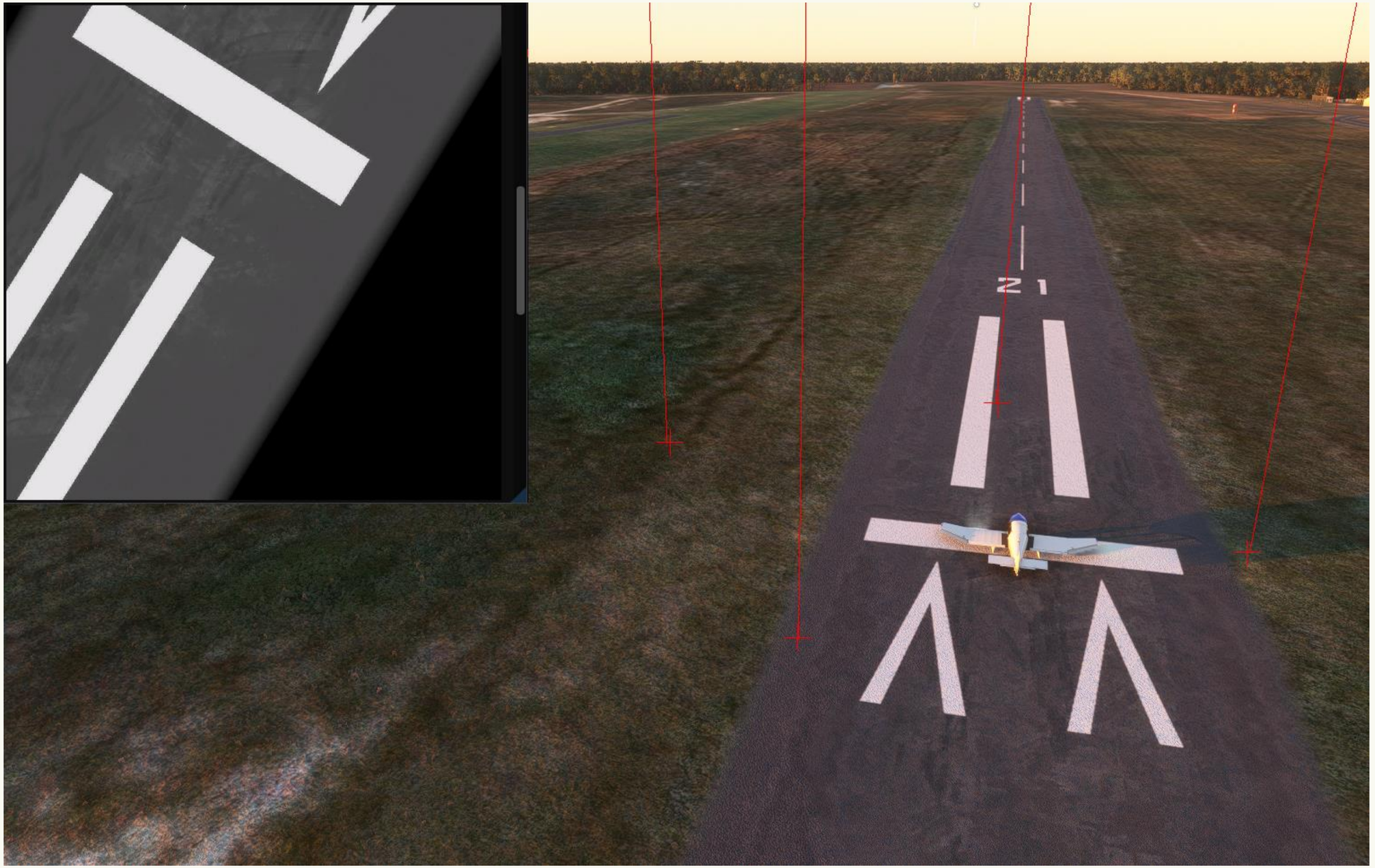
DEMDataMarker
DEMDataMarkerHeader
SurfaceCoverageLodRange
TerrainCoverageLodRange
SurfaceModelCoverageMapHeaderMarker
SurfaceModelCoverageMap
TerrainModelCoverageMap
GenIdManagerMarker
TerraformingMarker
WaterDataMarker
VectorHeaderMarker
VectorDataMarker
AdditionVectorHeaderMarker
AdditionVectorDataMarker
GameRender
VegetationMaskHeader
VegetationMask
MeshTex
ColorReferenceHeaderMarker
ColorReferenceMarker
HorizontalBreakReductionMarker
BingAerialHeaderMarker
BingAerialTex
BingTinMeshTex
GPSAerialTex
BingDemMesh
NoShadowCoverageMap
DataAugmentationCoverageMap
VegetationDetailMesh
VegetationMesh
WaterMesh
TinMaskHeaderMarker
TinMaskMarker
SecondaryAerialHeaderMarker
SecondaryAerialMarker
ColorMeanMarker
WaterBitmapHeaderMarker
WaterBitmapMarker
WaterBlurMaskHeader
WaterBlurMask
VegetationCollisionMarker
DataLayerHeaderMarker
DataLayerMarker
DataLayerLC30FullHeaderMarker
DataLayerLC30FullMarker
DataLayerRegionsHeaderMarker
DataLayerRegionsMarker
SynthesisMaskMarker
TINCCHeaderMarker
TINCCMarker
Meta
BlendedAerialMarker
EarthLightsHeaderMarker
EarthLightsMarker
SynthesizedAerialTex
CCActivationMaskMarker
BingSchemaMarker
BingOdvsChunkMarker
BingGridMarker
CLBMarker
WaterAlbedoHeaderMarker
WaterAlbedoMarker



Airports/airfields ground edition

- Requirements:
  - Modifiable at runtime (edition)
  - Support sloped runways
  - Support high-resolution textures (seen from up close)
  - Integrate with legacy FSX code
    - Legacy code generates and renders flat geometry
    - Uses a different subdivision scheme (not exactly a quadtree)!
- Solution:
  - Introduce the “GameRender” marker
  - Per-tile textures projected on the terrain

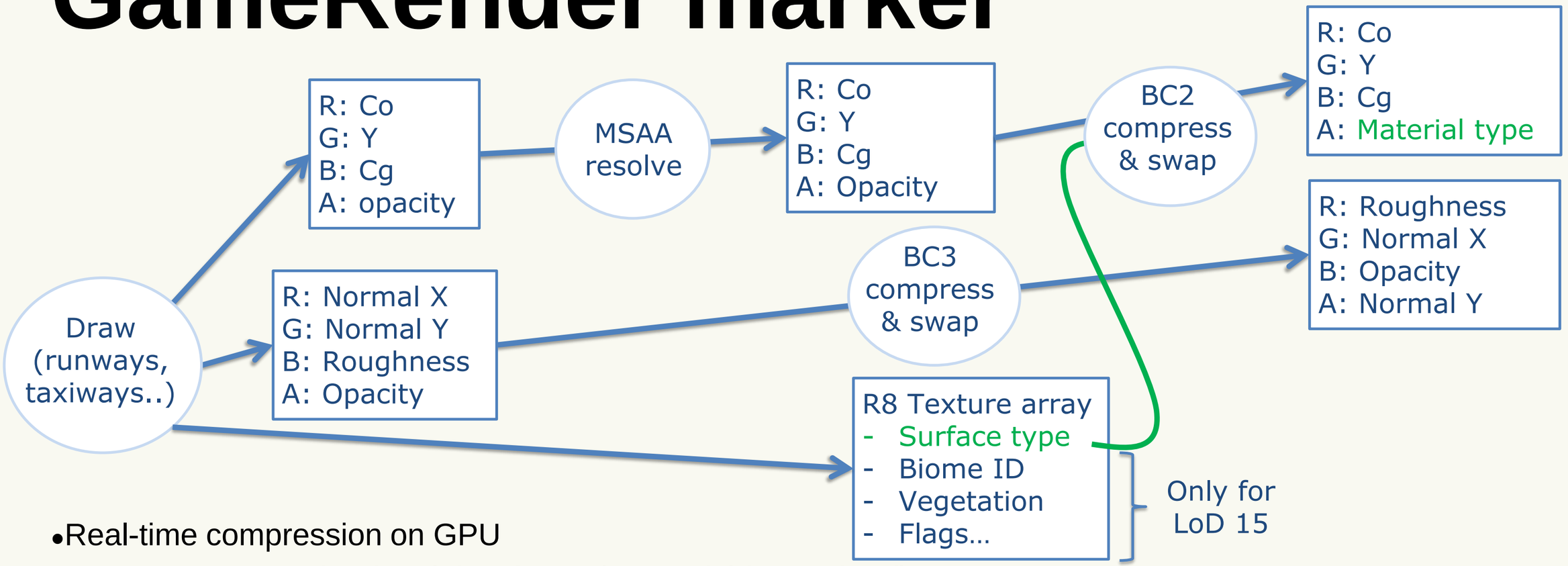



21

GameRender marker

- GameRender marker independent from aerial texture ⇒ resolution not impacted by aerial resolution
- Generated on-the-fly at the beginning of the frame
- Need to be at high quality
  - Antialiasing
  - Compression
  - Detect and avoid storing empty textures
- LoD 15 is read back to CPU
  - Collisions
  - Trees/grass generation

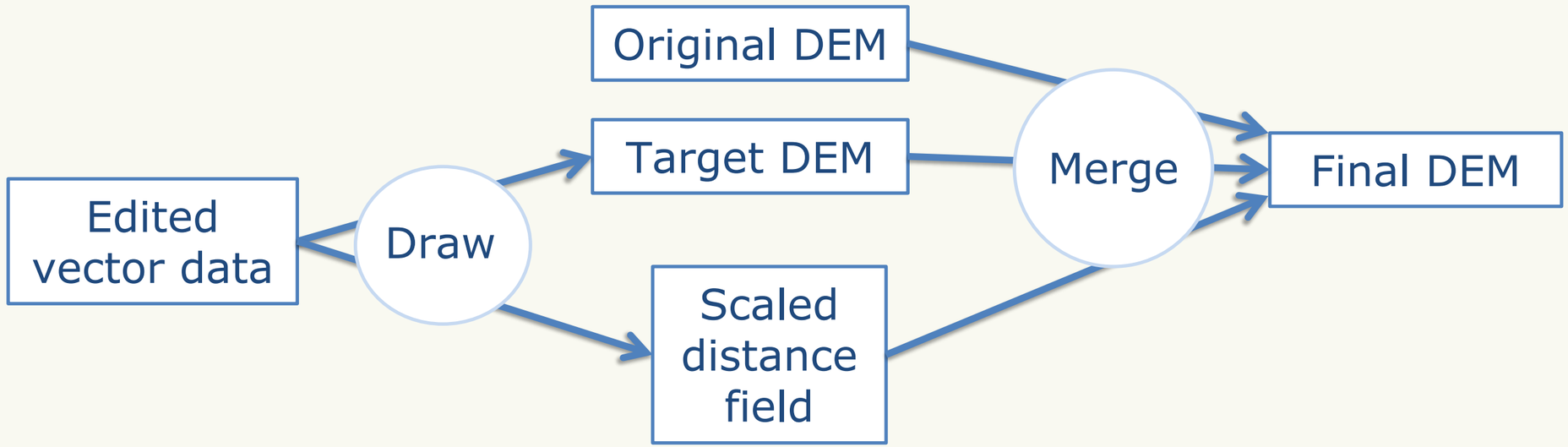
GameRender marker



- Real-time compression on GPU
- BC2: no compression for alpha channel ⇒ useful for storing flags or enum values
- YCoCg reduces color shift (“gray becomes purple” syndrome)
  - Store Y in green channel (6 bits precision)
  - Co and Cg have the same precision (5 bits)
- Use visibility queries to detect empty textures in the next frame. If empty, return textures to pool.

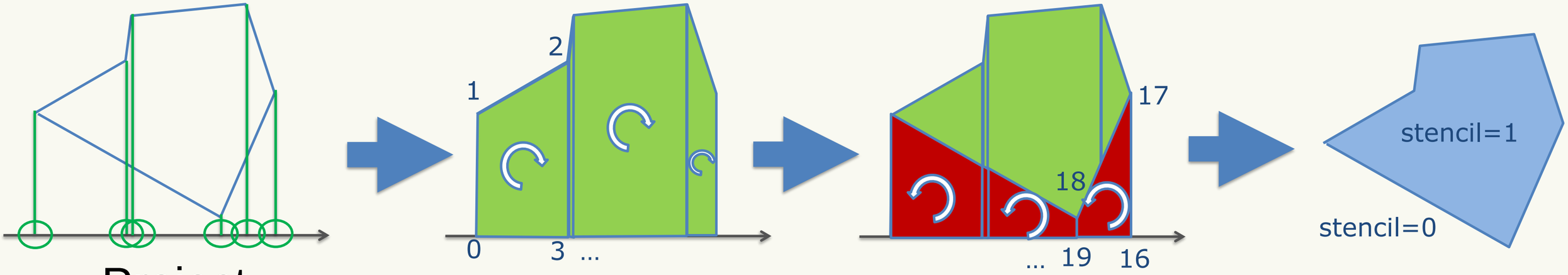
Terraforming

- Edition needs to be compatible with future DEM updates
  ⇒ Avoid editing original DEM data
- Edit vector data: points, rectangles and polygons + heights + influence radius



Distance field generation

1) Filling the interior



Project vertices to horizontal axis

Draw to stencil buffer:

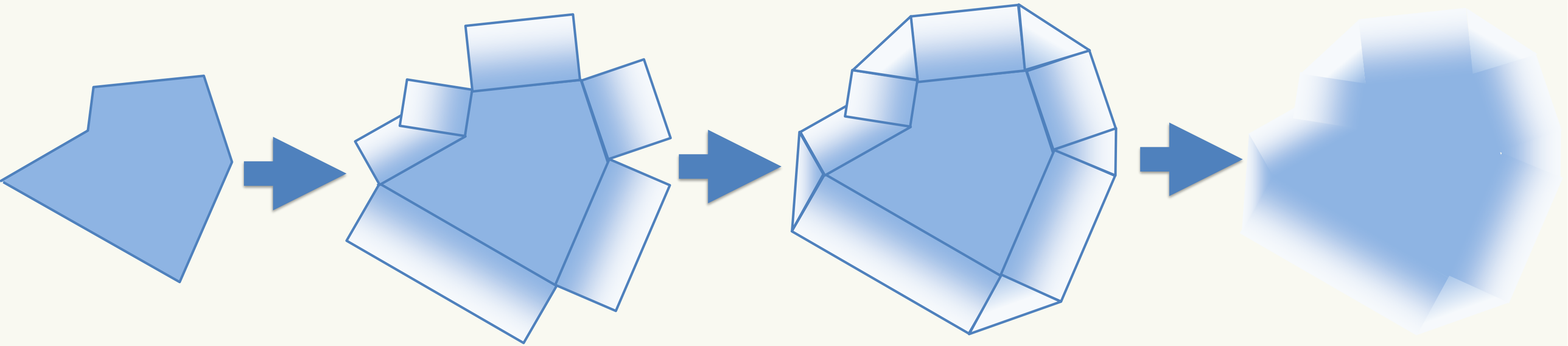
- `INC` when `CW`
- `DEC` when `CCW`

Draw quad with stencil test `COMPARE_EQUAL`

Distance field generation

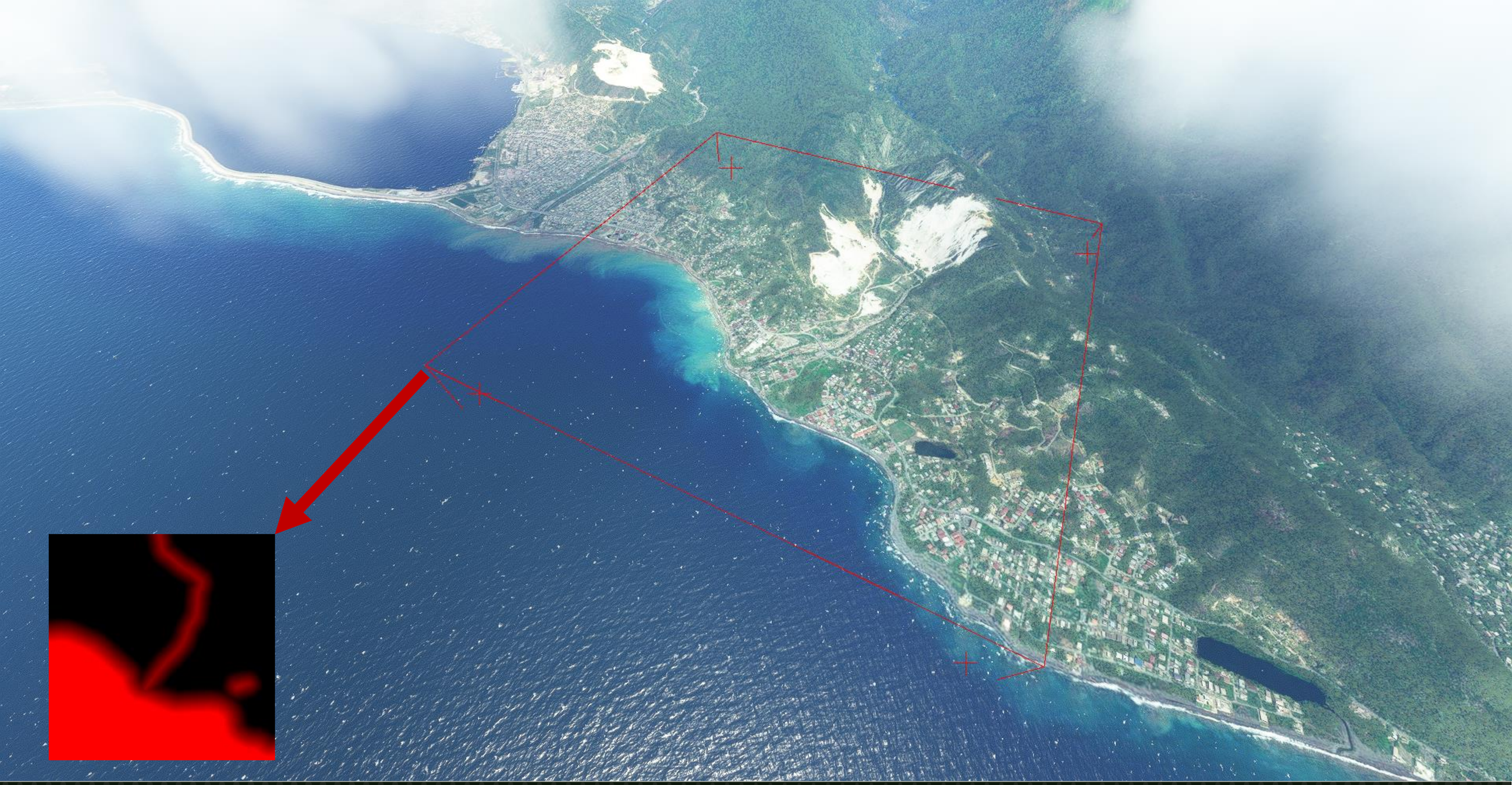
2) Filling the borders

- Extrude polygon
- Vertex interpolation
- Stencil test to avoid overriding the interior
- Use `BLEND_OP_MIN` to handle overlap



Water: rendering

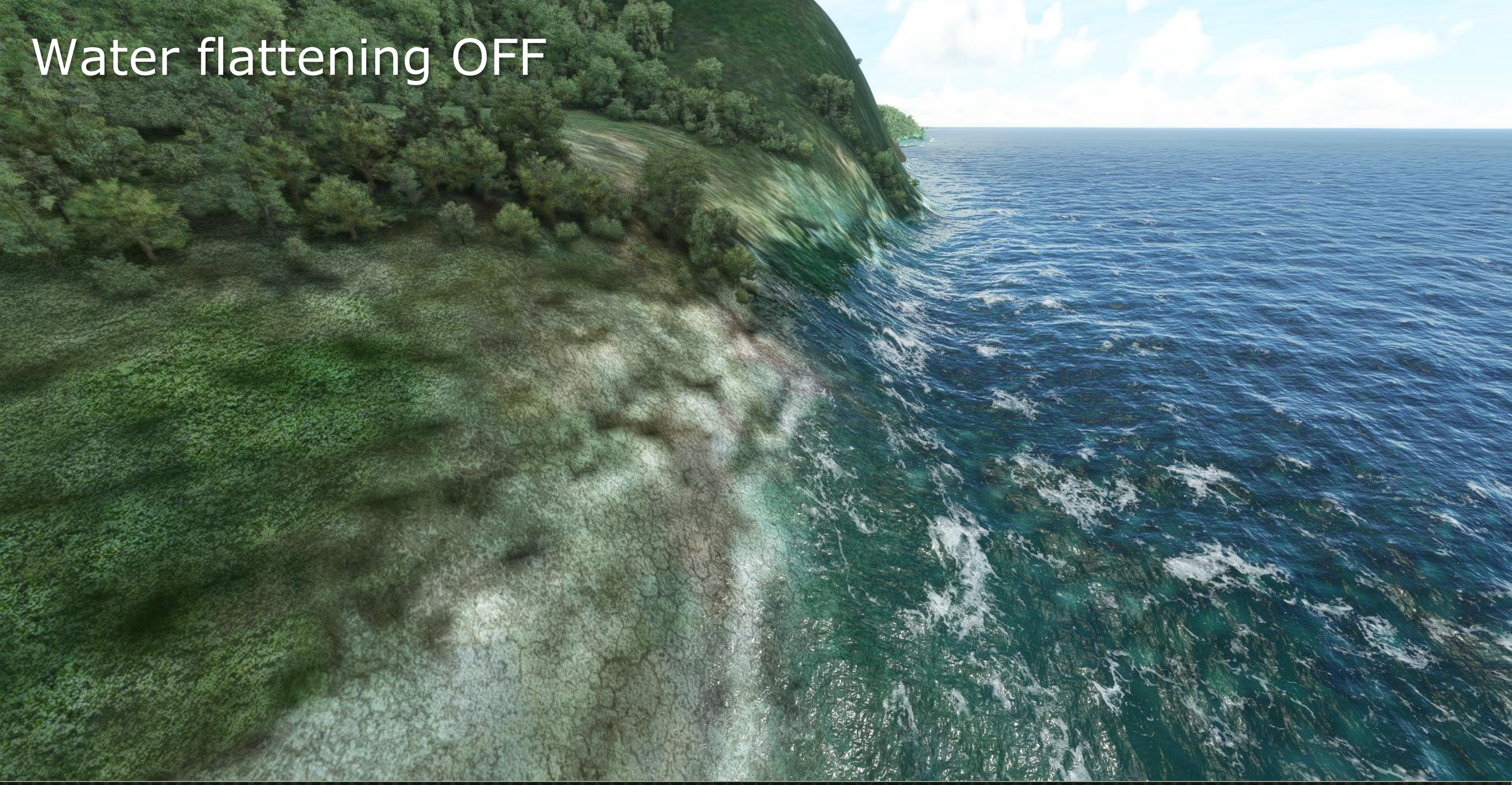
- Drawn using terrain geometry
  - Pixel shader branching
- Generate distance field from vector data (same method)
- Use distance field for
  - Blending with aerial
  - Water flow and foam (gradient)
  - Flattening
- Hand-edited « blur » mask



Water: flattening

- Similar to terraforming
- Challenges:
  - TIN: low tessellation level
    ⇒ Limited vertex displacement
  - DEM: flat water VS hill
    ⇒ some popping when LoD changes
    ⇒ adjust flattening strength according to height difference

Water flattening OFF



Water flattening ON

Vector data: flexibility

- Vector data:
  - Resolution-independent
  - Avoids changing the original data
- Rasterizing vector data to per-tile textures
  - Projects to the terrain ⇒ supports arbitrary relief!
  - Can generate distance field on GPU for smooth fade-outs
- Using vector data for:
  - Drawing on top of aerial data
  - Terraforming
  - Water
  - Flags and scales: trees, grass, excluding buildings…


B) Make a terrain system that scales

Augmentation types

	Runtime augmentations	Pre-computed augmentations
Iteration time	Instantly applied on the entire Earth	Heavy processes running in the cloud
Available data	Limited: only the currently streamed data available	Can access any tile at any LOD
Compute power and memory limits	Limited by the client hardware	Limited by resources
Examples	- Grass detection - Detail maps - Precise trees positioning	- Color correction - Vegetation detection - Image quality evaluation

Scale with hardware specs

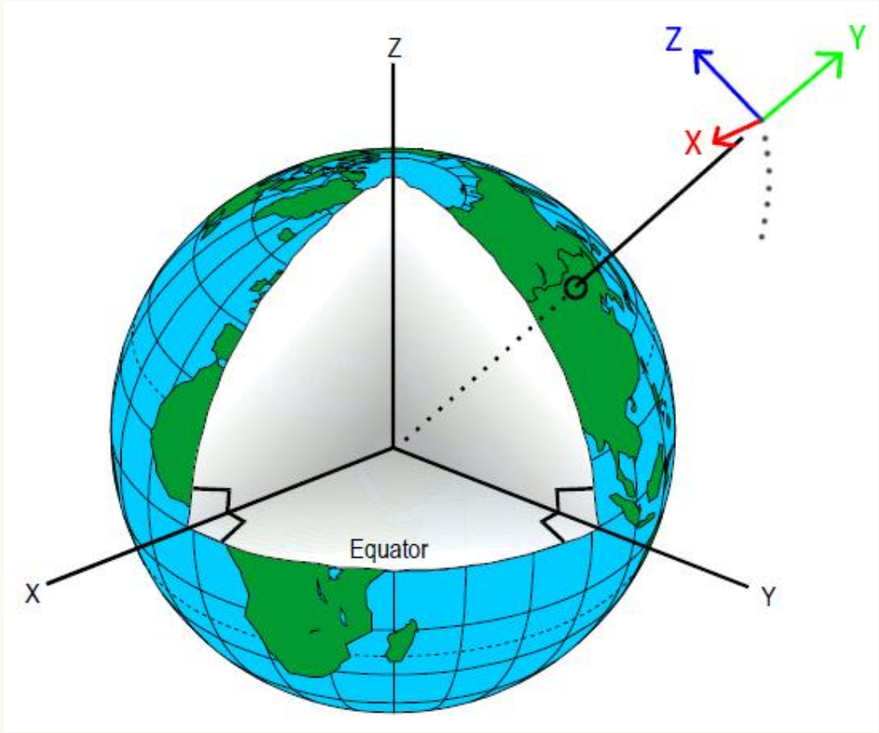
- Quadtree: good fit for scaling with hardware specs
  - Just scale the LoD choice factor
  - Using a fixed LoD for some data types is problematic in theory
    - E.g. trees available at LoD 13 & 15
    - Uneven distribution (Mercator projection)
    - Not that much an issue in practice
- Varying network bandwidth
  - Dynamically measure bandwidth
  - Continuously correct requested LoDs

Handling large coordinates

- CPU node position
  - Float 32 bits → Double 64 bits
- GPU:
  - Inverted depth buffer
  - Introduce « anchor space »

Anchor space

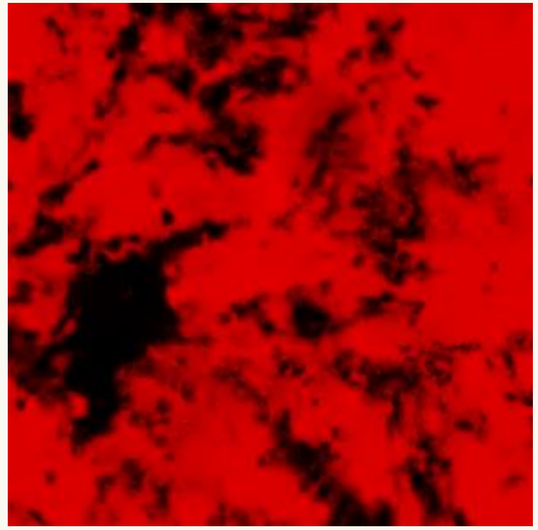
- Origin = camera position
- Rotation:
  - Y = Earth surface normal
  - X and Z
    - Computed from previous frame
    - Drift over time



- Before: `localPos → worldPos → viewPos → projPos`
- After: `localPos → anchorPos → viewPos → projPos`


Trees!
23 M generated trees
6.3 M trees in frustum

Trees



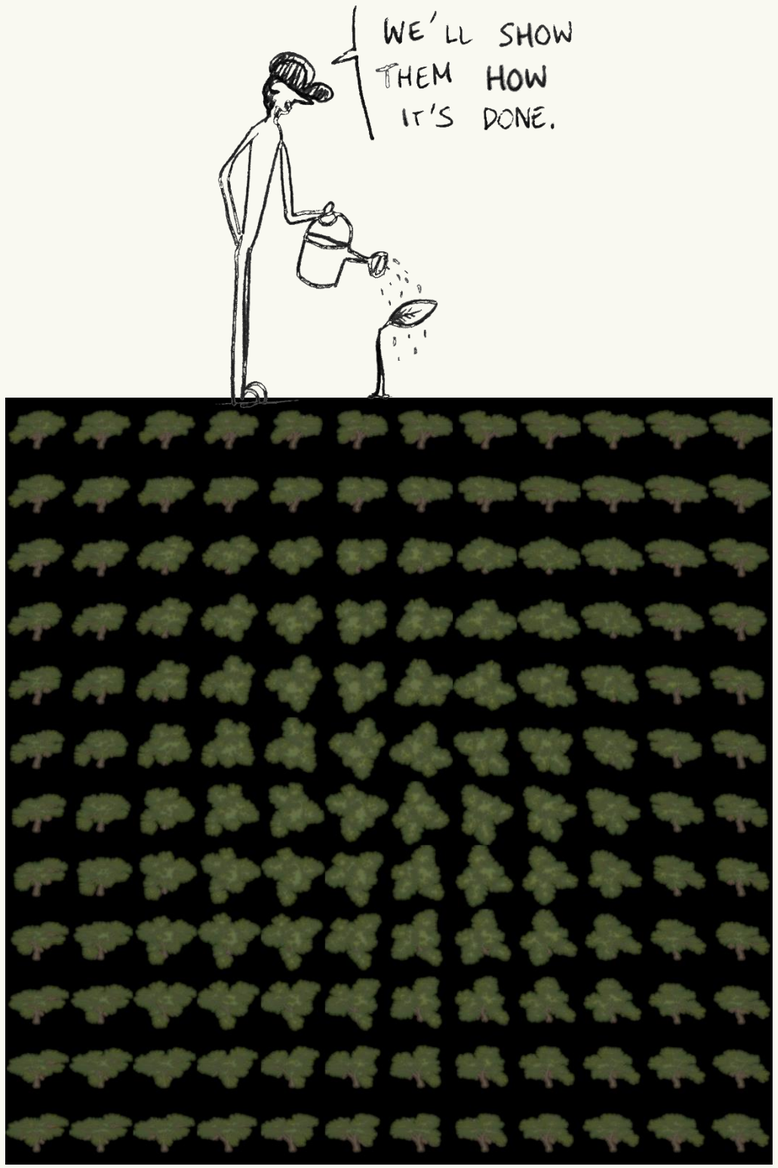
- Per-tile vegetation marker (LoD 13 & 15)
- Per-tile trees mask
  - Pre-computed in Azure using machine learning
- Use Halton sampling
  - Always uniform, whatever the chosen density
  - Density can be controlled at draw time
- Tree types determined using rules based on biome data
- CPU: sample DEM ⇒ set correct height
- CPU: sample aerial ⇒ set matching tree color

  ⇒ Trees fade out naturally to the aerial!

Trees

- Rendering
  - Display millions of trees ⇒ 3D imposters [2]
  - 1 draw call per tree specie per tile
  - Vertex buffer: 1 vertex per tree
- TIN
  - Baked-in TIN trees not good enough
  - Flattening at runtime problematic (LoD popping)

  ⇒ Draw imposters on top of TIN geometry



TIN area: Trees OFF

TIN area: Trees ON

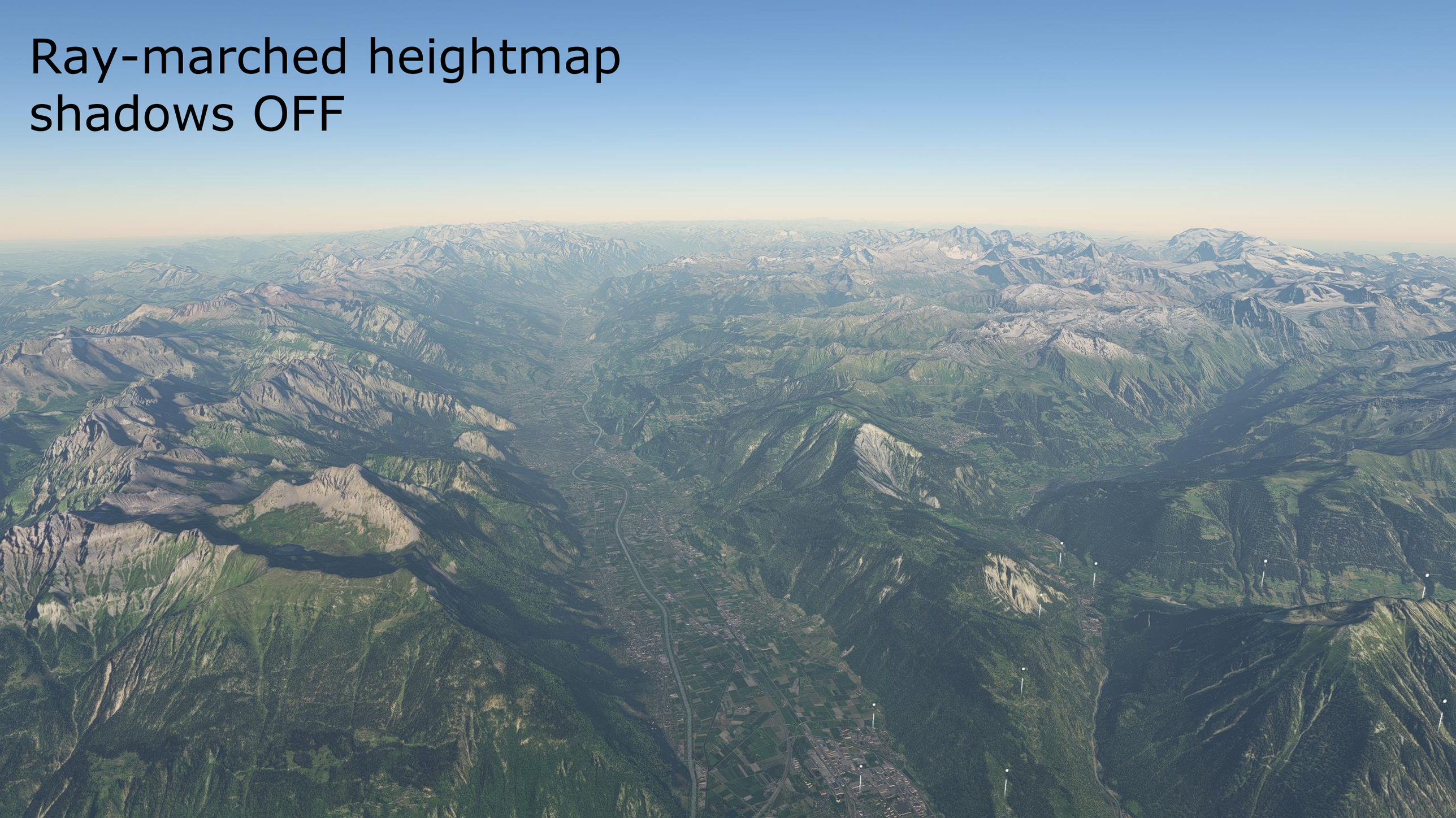
DEM+aerials area: Trees OFF

DEM+aerials area: Trees ON

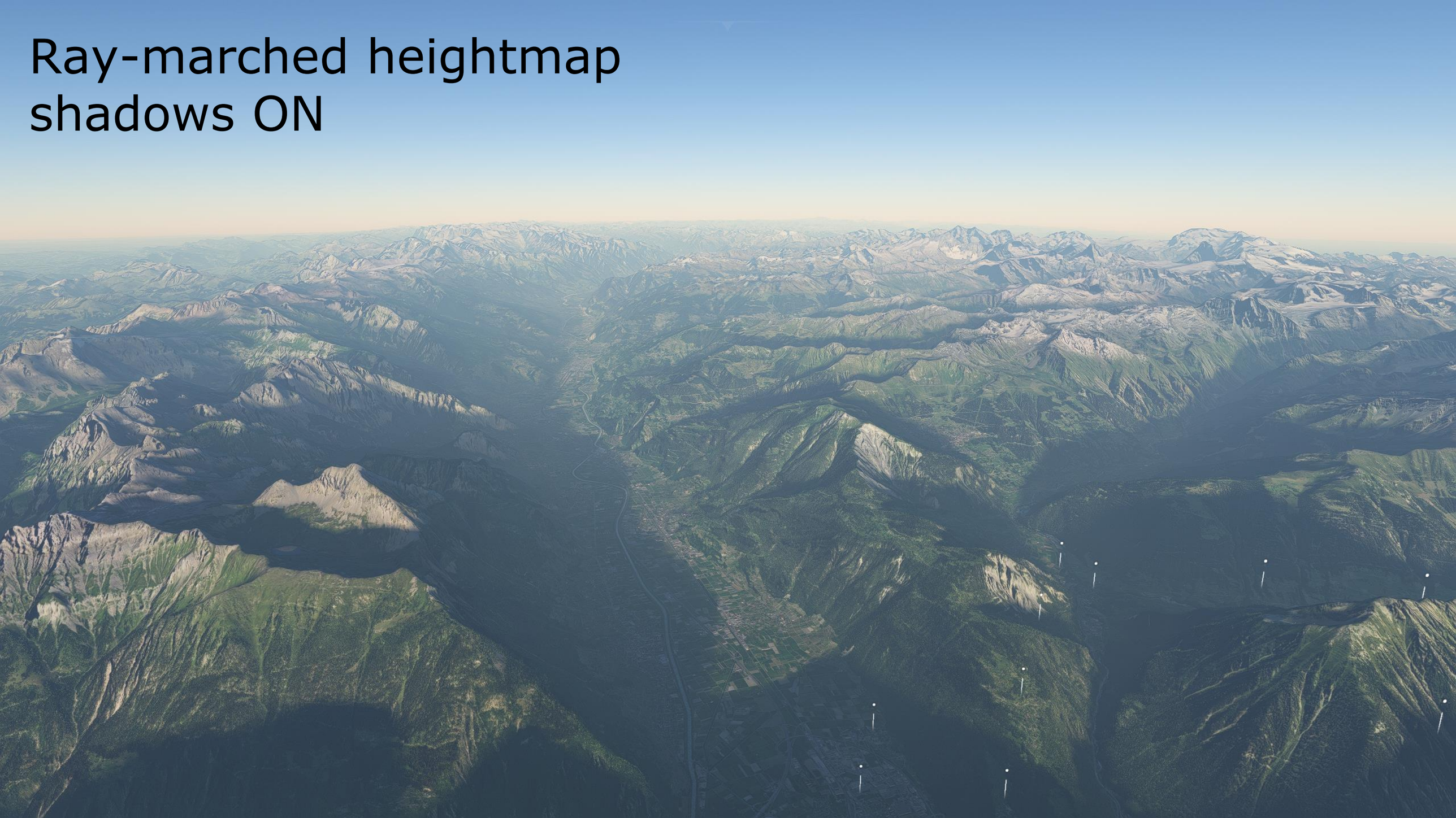
Shadows

- Combination of techniques:
  - Cascaded shadow maps
    - First slice fits the plane/cockpit
    - Limited in distance
  - Big terrain shadows (mountains…)
    - Top-down render of the terrain to a heightmap
    - Ray-marching inside the heightmap
    - 300 km x 300 km
  - Small shadows on screen
    - Screen space ray-marched shadows

Ray-marched heightmap shadows OFF



Ray-marched heightmap shadows ON




Screen-space ray-marched
shadows OFF


Screen-space ray-marched
shadows ON

Ground details

- Detail maps
  - Near, mid and far
  - Dirt / Asphalt / Grass
  - Runtime detection in pixel shader
- Grass
  - Spawned according to surface type
  - Colored with aerial


Detail maps OFF
Grass OFF


Detail maps ON
Grass OFF

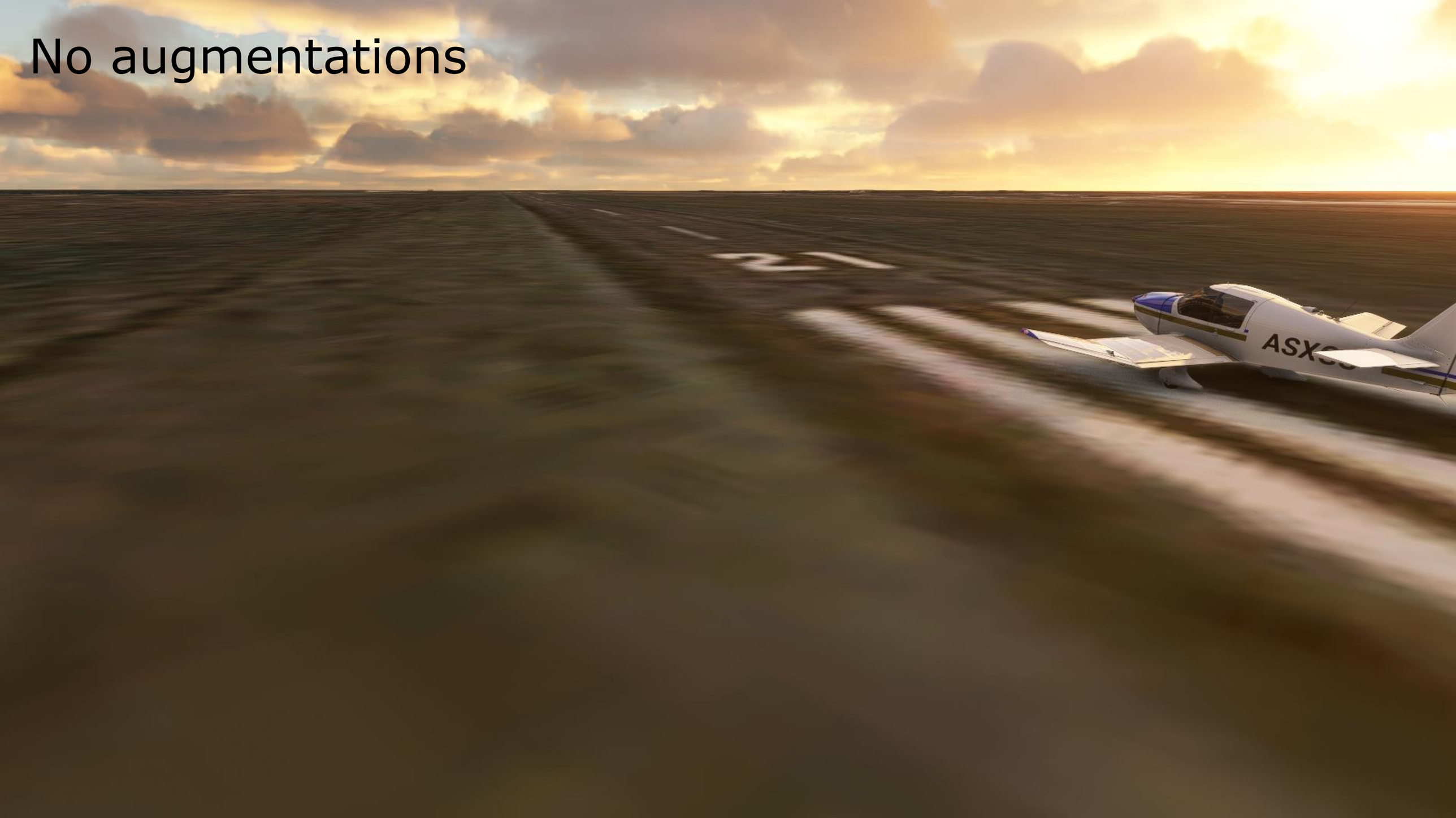

Detail maps ON
Grass ON


Detail maps OFF
Grass OFF


Detail maps ON
Grass OFF


Detail maps ON
Grass ON

No augmentations



Buildings

Buildings + Trees

Buildings + Trees + GameRender


Buildings + Trees + GameRender
+ Detail maps

Buildings + Trees + GameRender
+ Detail maps + Grass


We are hiring ☺

THANKS

- Thank you for listening
- A big thank you to all the Flight Simulator team
  - Special thanks to the Asobo Engine team, you rock!

- Questions?

References

[1] https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system

[2] https://shaderbits.com/blog/octahedral-impostors