BroadLeaf:

Real-Time Cinematic Rendering of Large-Scale Forests

Yuewei Shao[1], Yixin Hu[2]

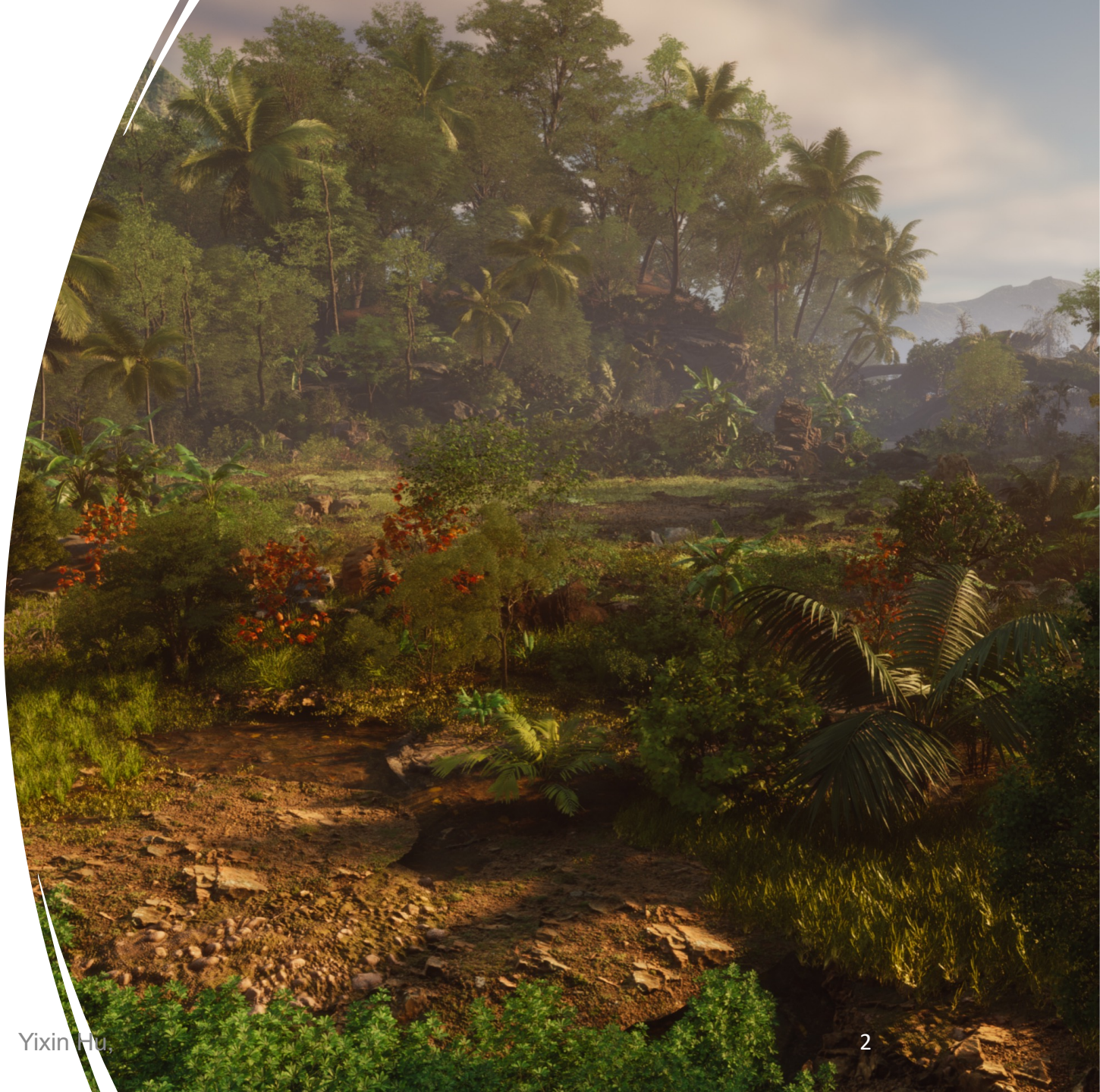
[1] START Cloud Gaming, Tencent

[2] Pixel Lab, Tencent America

Tencent Games

PART I: Introduction

Problem Definition & Existing Methods



Introduction

- **The problem to solve.**
  - Rendering large-scale high-quality trees in real-time, preferably interactable.
- **Why is the problem difficult?**
  - Complex geometry and texture
  - Data storage burden
  - Data streaming burden

Existing Solutions

Limitation of the state-of-the-art solutions (LOD based):

- Manual generated LOD, but not automatic.
- LOD transition is not smooth or fast.
- Not suitable for complex and large-scale forest scenes.

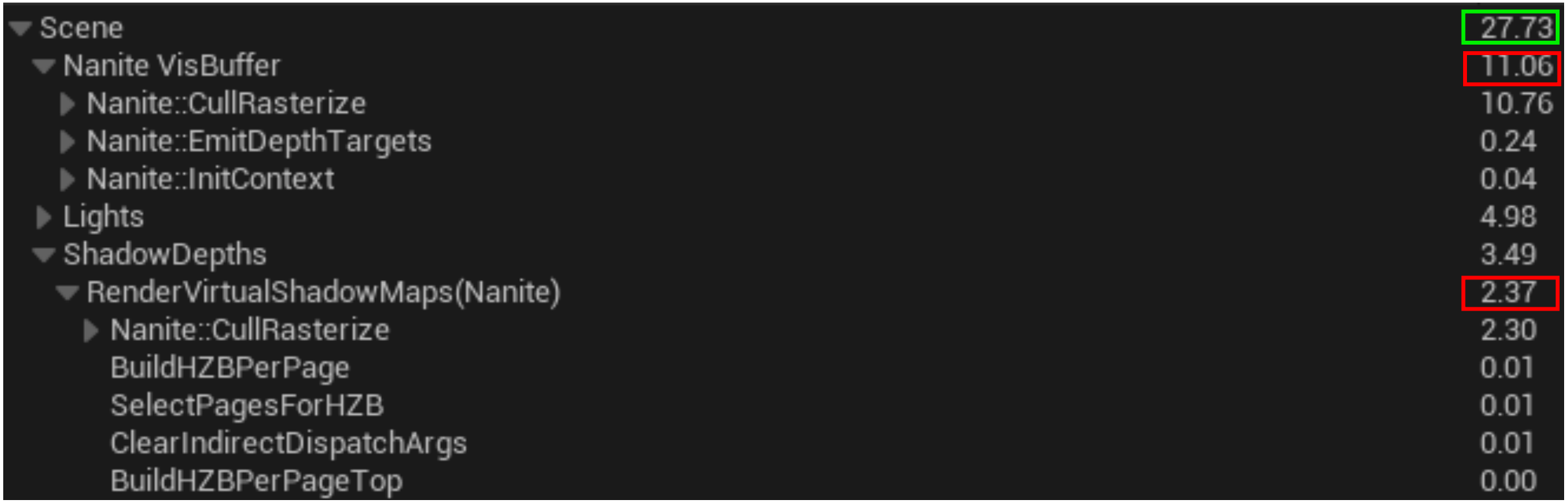
Existing Solutions (1)

Over-simplification issue for thin surfaces like grass and leaves.


~600 million triangle faces

Existing Solutions (2)

Nanite Foliage performance on our demo scene:



Our Goal

Expected solution:

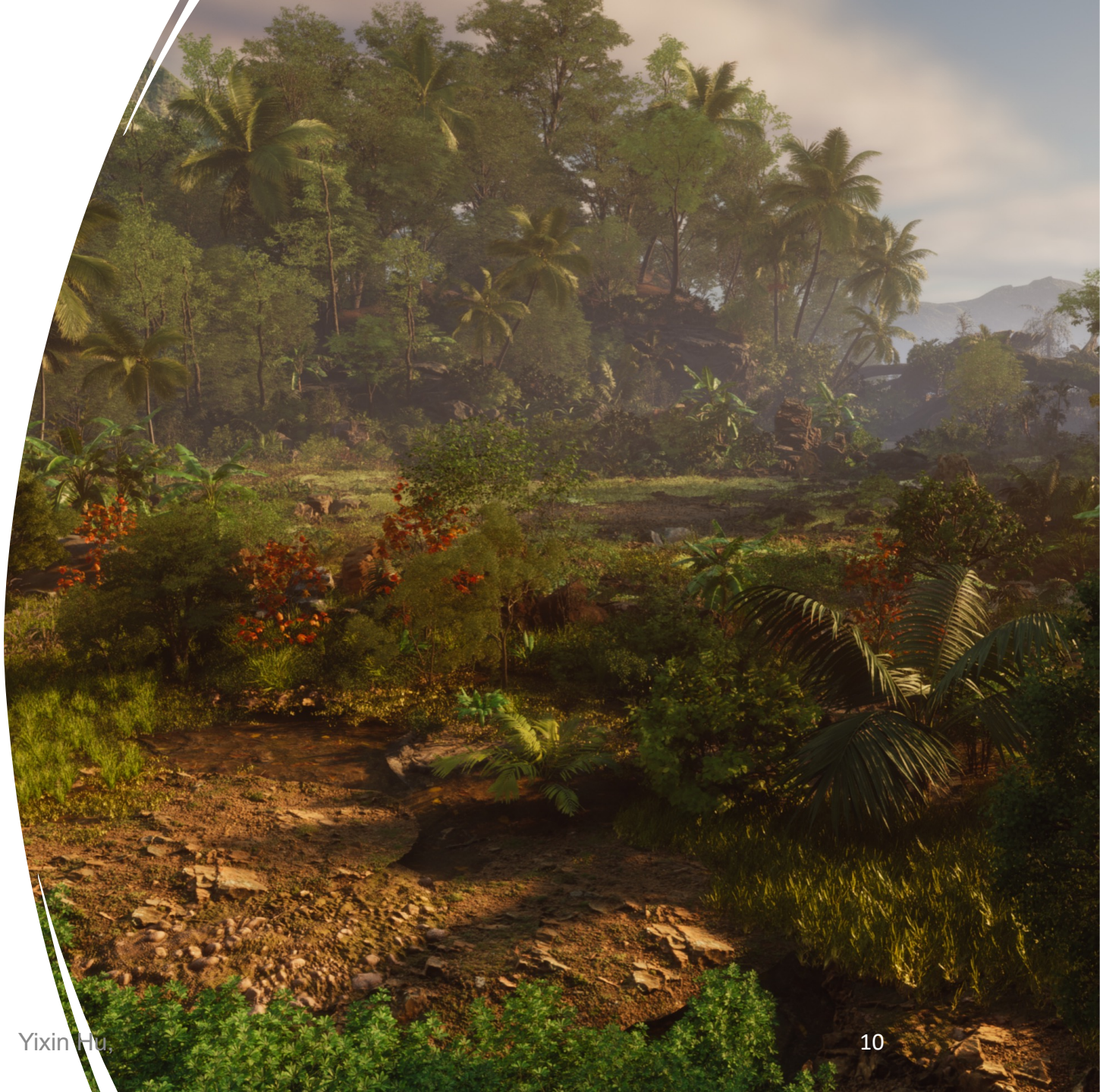
- Scene: Large-scale; real-time; cinematic quality (minimum quality loss when improving the efficiency); interactable.
- LOD for tree leaves: Automatic, smooth, and fast.
- Our application environment: Cloud gaming with powerful GPUs.


Powered by BroadLeaf

PART II:

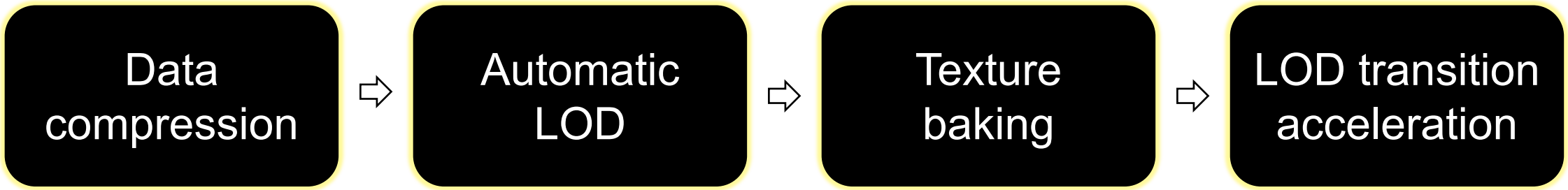
BroadLeaf

Algorithm & Technical Details



Algorithm Overview

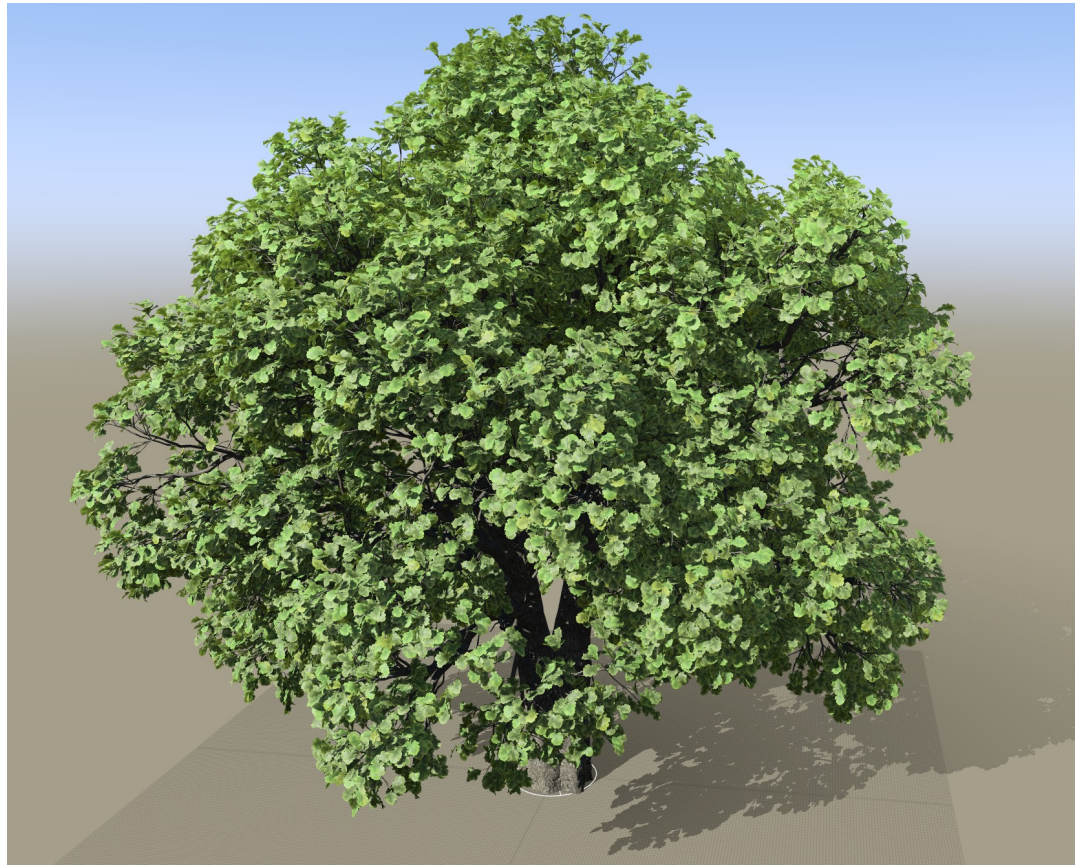
BroadLeaf algorithm pipeline:



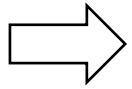
Data Compression (1)

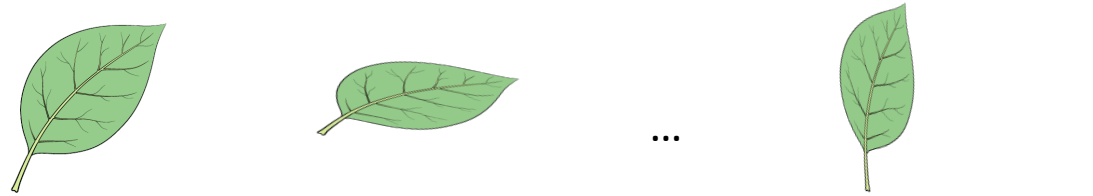
A high-quality tree model:

~20 million faces, 500 MB



Data Compression (2)





Base leaf data:

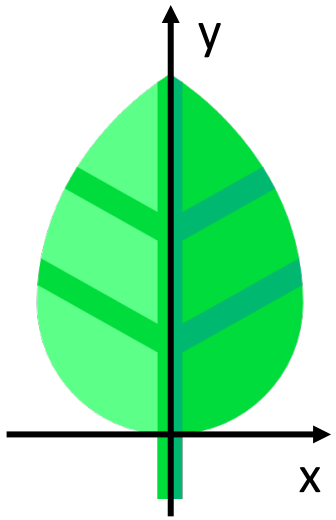
- Geometry: Vertices, faces, and UV coordinates.
- Transformation: translation, uniform scaling, and rotation.
- Texture maps.

Data Compression (3)

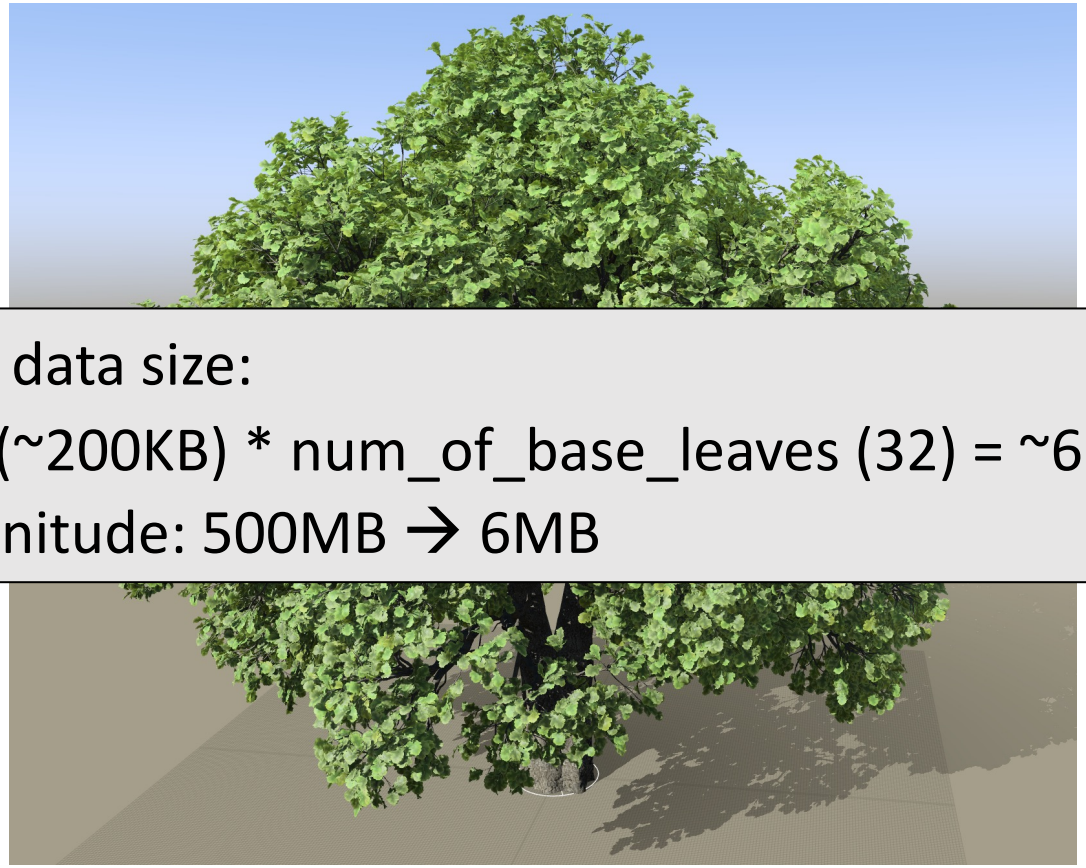
- Trees without reference information:
  - Usually from modeling tools, e.g. SpeedTree.
- Trees without reference info:
  - Pre-categorize by materials.
  - Categorize by UV coordinates.
  - Calculate transformations.
  - Choose several arbitrary curling effects for each category.

Data Compression (3)

- Leaf internal deformation:
  - Fold: Folds both sides of the leaf along the Y-axis.
  - Curl: Curls the leaf around the X-axis
  - Twist: Twists the leaf around the Y-axis.



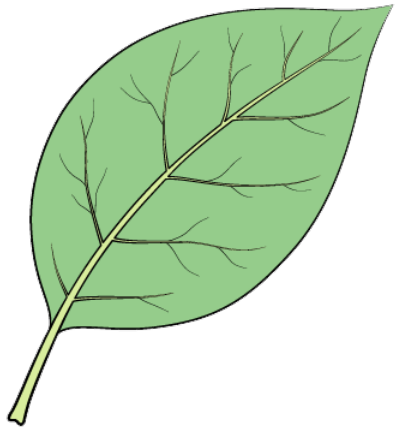
Data Compression (4)

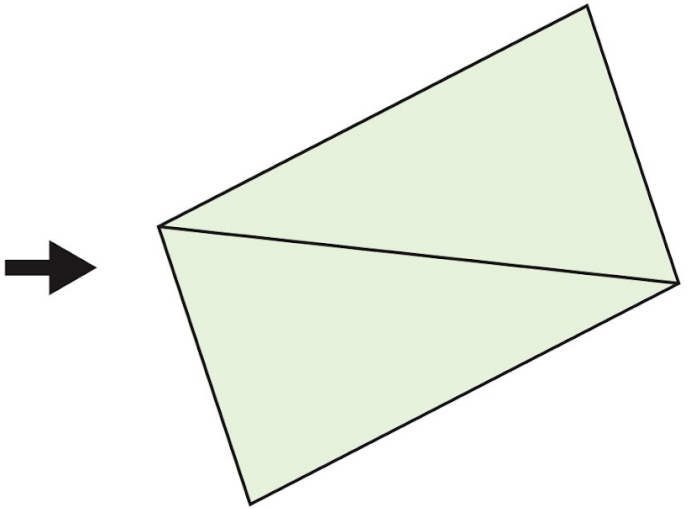


- Compressed geometry data size:
  - leaf_instance_size (~200KB) * num_of_base_leaves (32) = ~6.4MB
- Around 2 order of magnitude: 500MB → 6MB

Automatic LOD (1): Quad-leaf

Base leaf



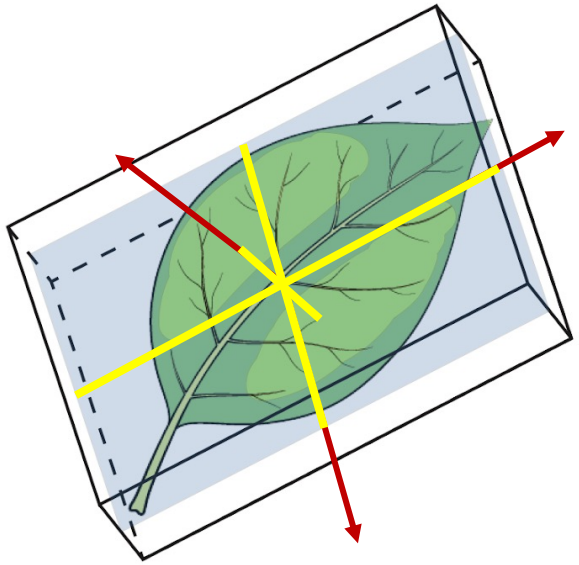


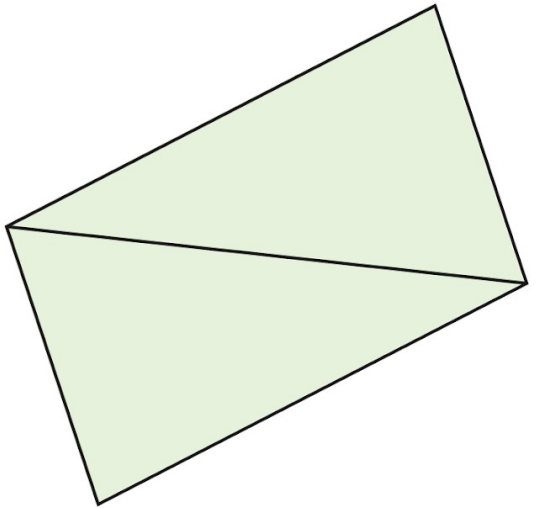
Quad-leaf

Automatic LOD (1): Quad-leaf

Object-oriented bounding box

Base leaf





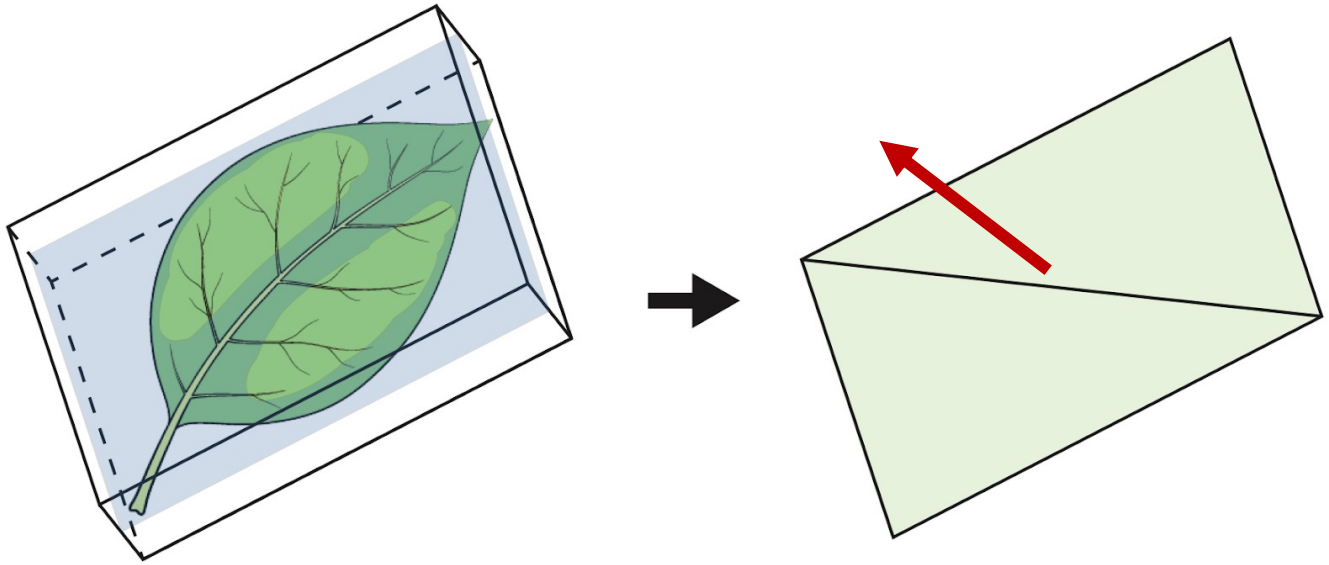
Quad-leaf

PCA: Compute the 3 principal axes.

Automatic LOD (1): Quad-leaf

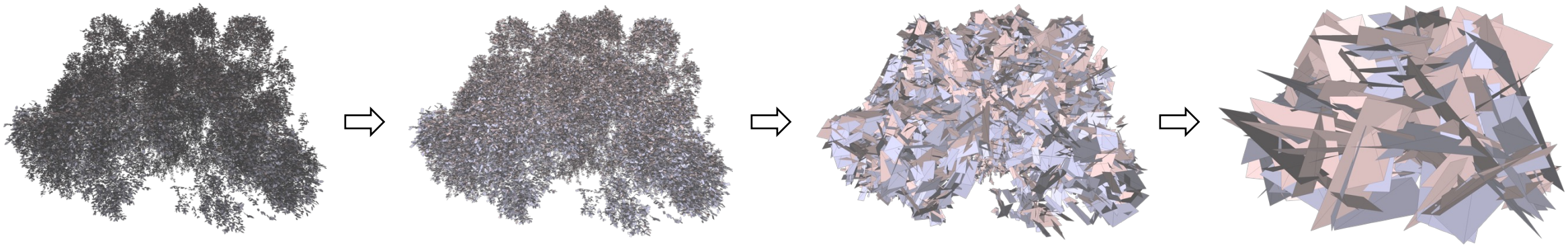
Object-oriented bounding box

Base leaf



Quad-leaf

Automatic LOD (2): Quad-leaf



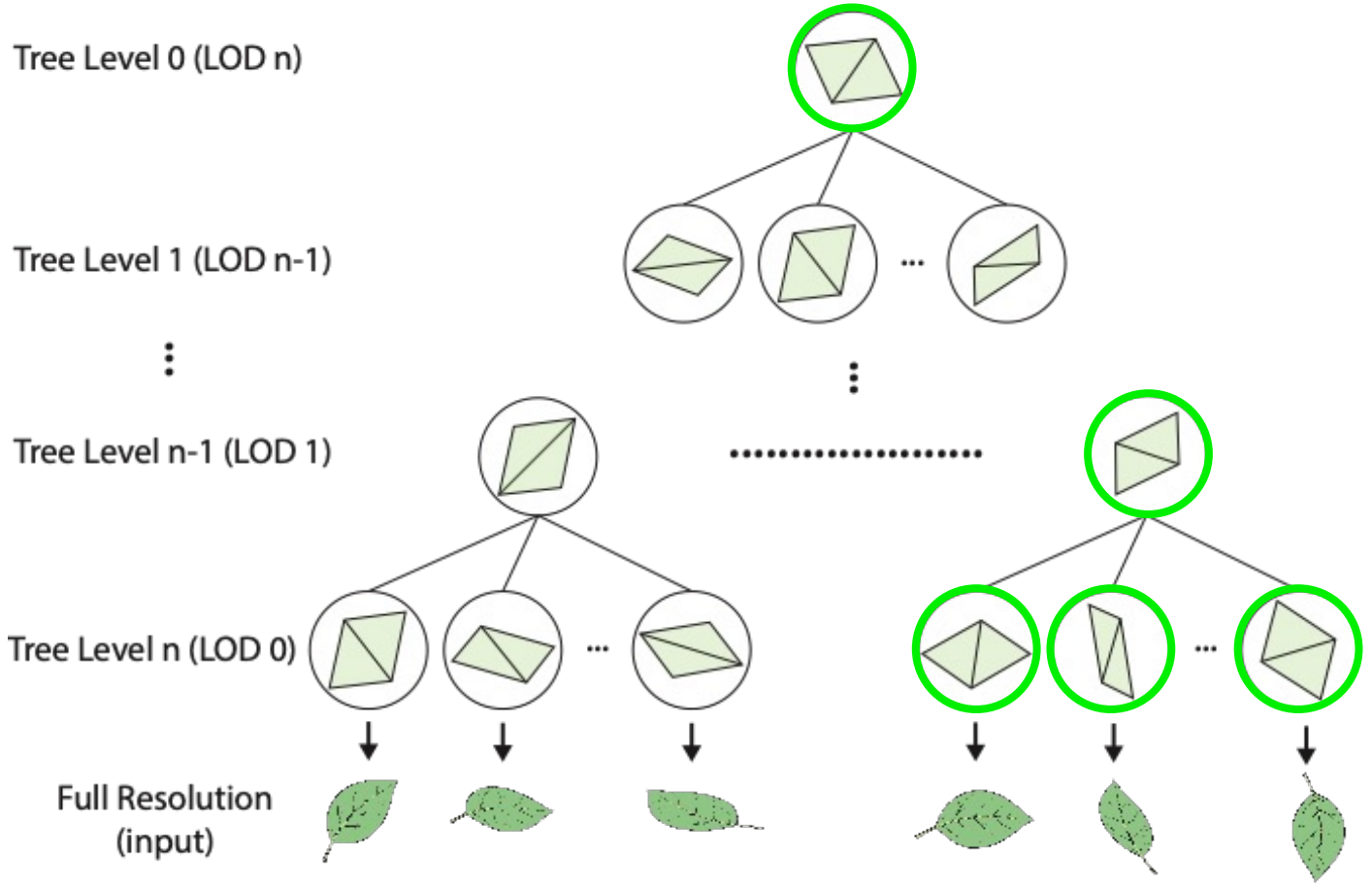
Input
#F: 6,004,752

LOD 0
#F: 307,936

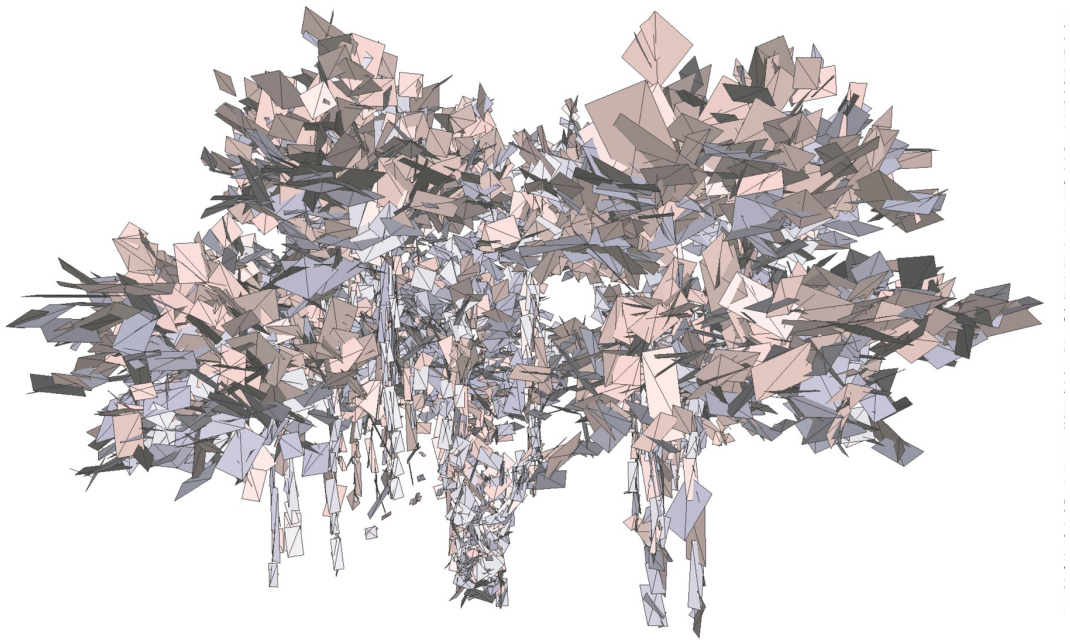
LOD 1
#F: 15,408

LOD 2
#F: 762

Automatic LOD (3): LOD Tree

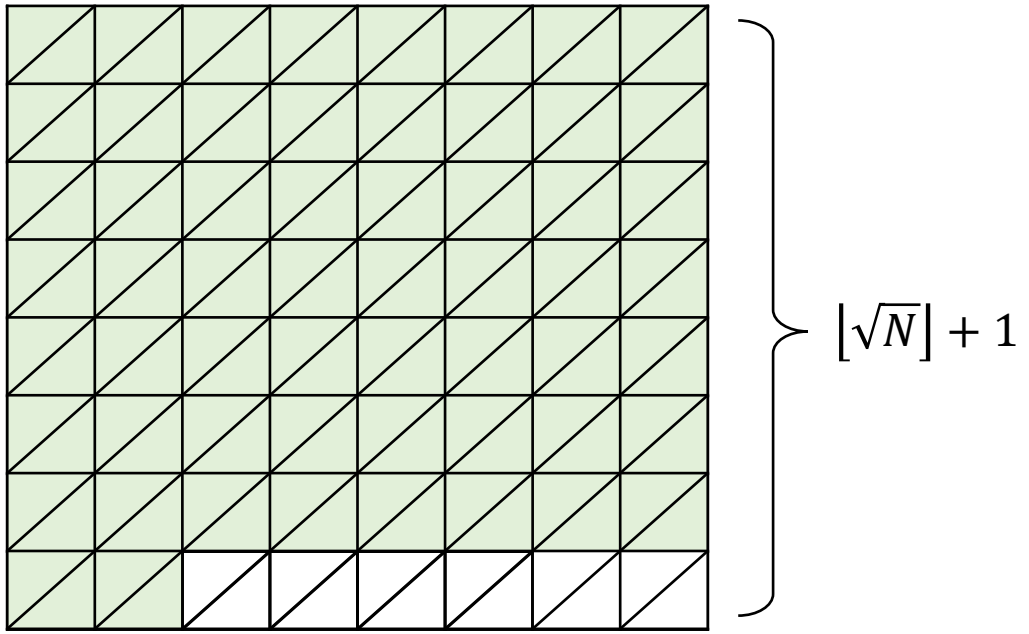


Texture Baking (1)



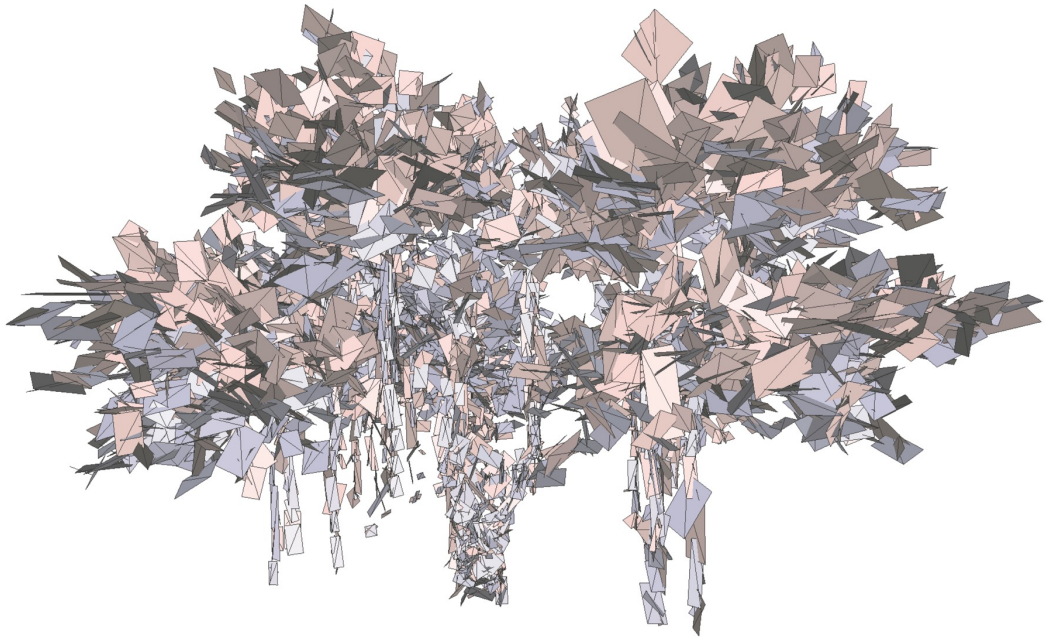
Mesh of *N* quad-leaves
#Face = 2*N*

#UV grids $M = (\lfloor\sqrt{N}\rfloor + 1)^2$

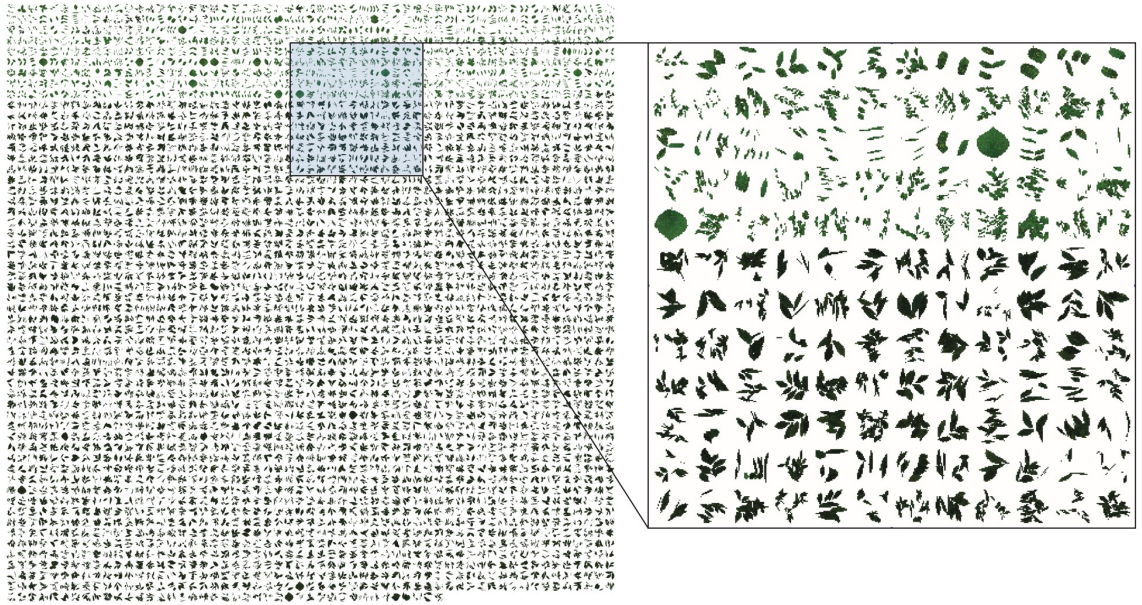


UV mapping

Texture Baking (2)



Mesh of *N* quad-leaves
#Face = 2*N*



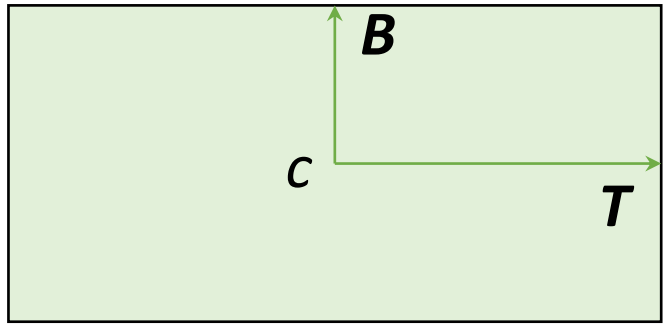
Albedo map of the mesh on the left

LOD Transition (1)

Requirements:

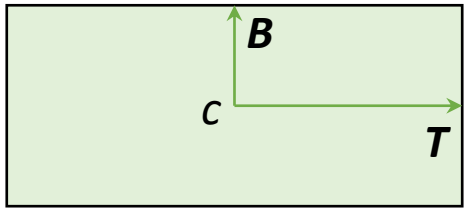
- Visually seamless transition
- Computational fast

A quad-leaf is represented as:

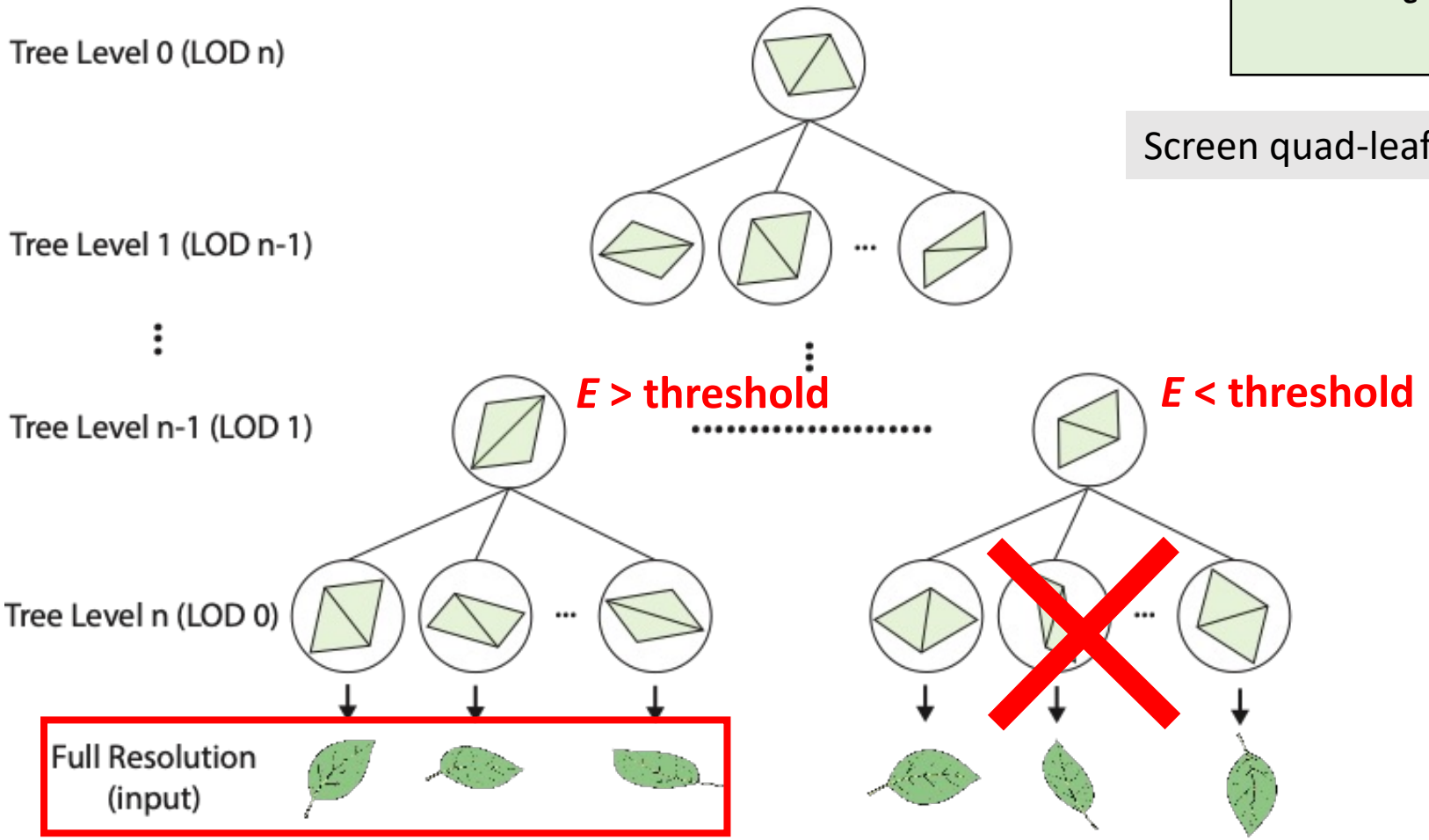


c: Center (float3)
T: Tangent vector (half3)
B: Binormal vector (half3)
c': UV coordinate (float2)
t: UV tangent length (half)
b: UV binormal length (half)

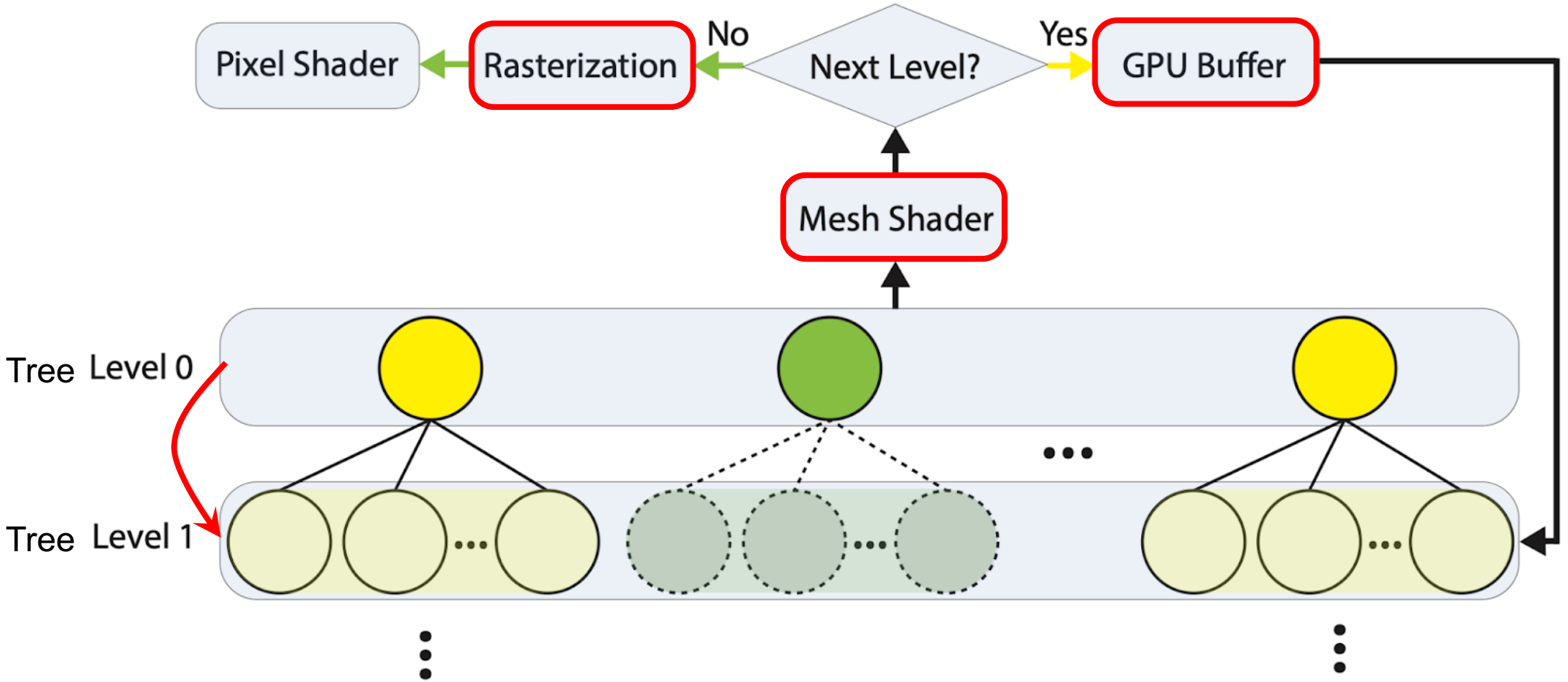
LOD Transition (2)



Screen quad-leaf size: $E = |\boldsymbol{B}|+|\boldsymbol{T}|$



LOD Transition (3)

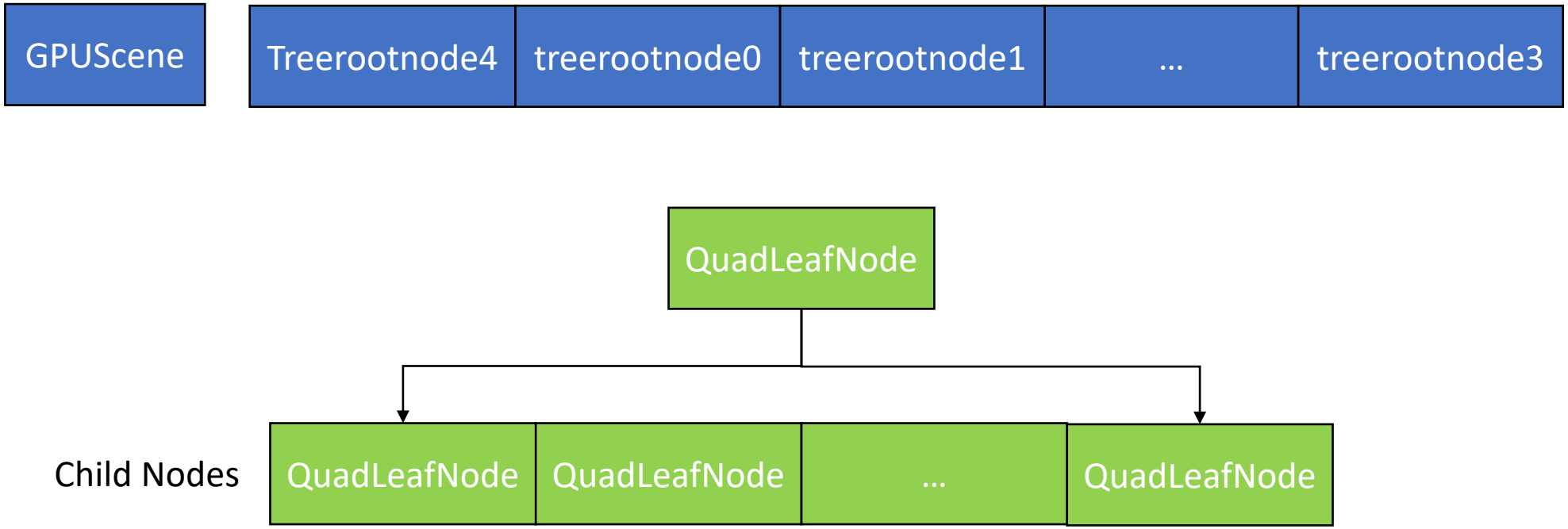


LOD Transition (4)

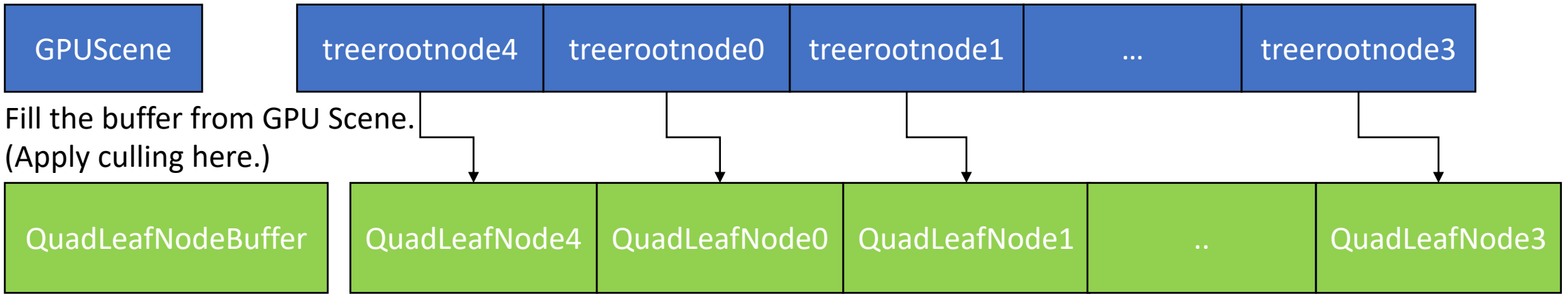
Runtime data structure for GPU driven:

```
Struct treerootnode{
    uint treenodetype;
    float3 center;
    float radius;
};
```

```
Struct QuadLeafNode{
    float3 centerpos;
    half3 tangent;
    half3 binormal;
    float2 uv;
    half2 uvextent;
    uint leafid;
    uint childnodestartindex;
    uint childnodecount;
};
```



LOD Transition (5)



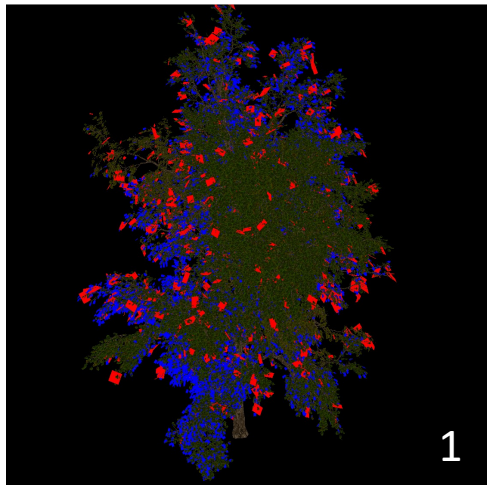
Dispatch the mesh shader according to the Quadleafnode Buffer count.

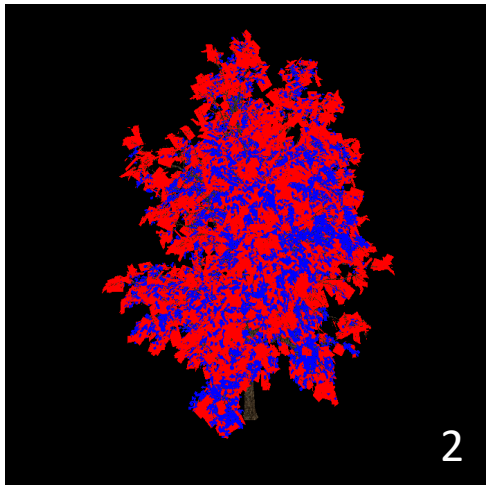
In mesh shader, we compute the screen size of quad-leaves.

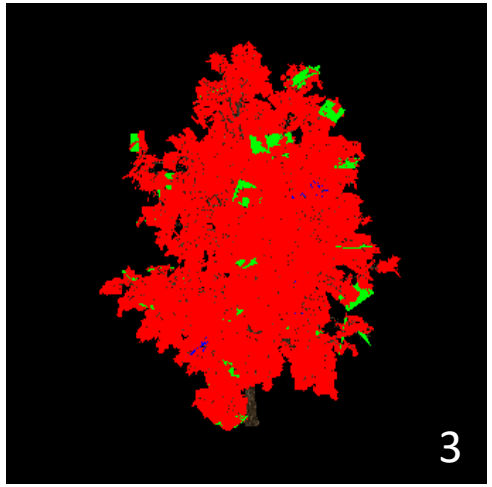
Why use mesh shader?

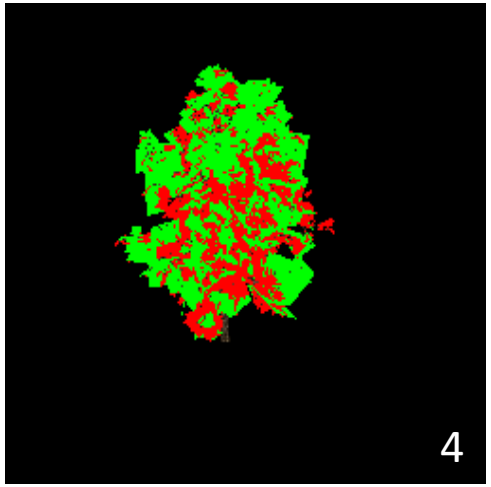
- We can traverse the details of the tree and draw the proper size of quad-leaves.
- If using computer shader, we have to write and read the quad-leaves that need to render to a buffer for rendering, which needs more GPU access bandwidth.

LOD Transition Result









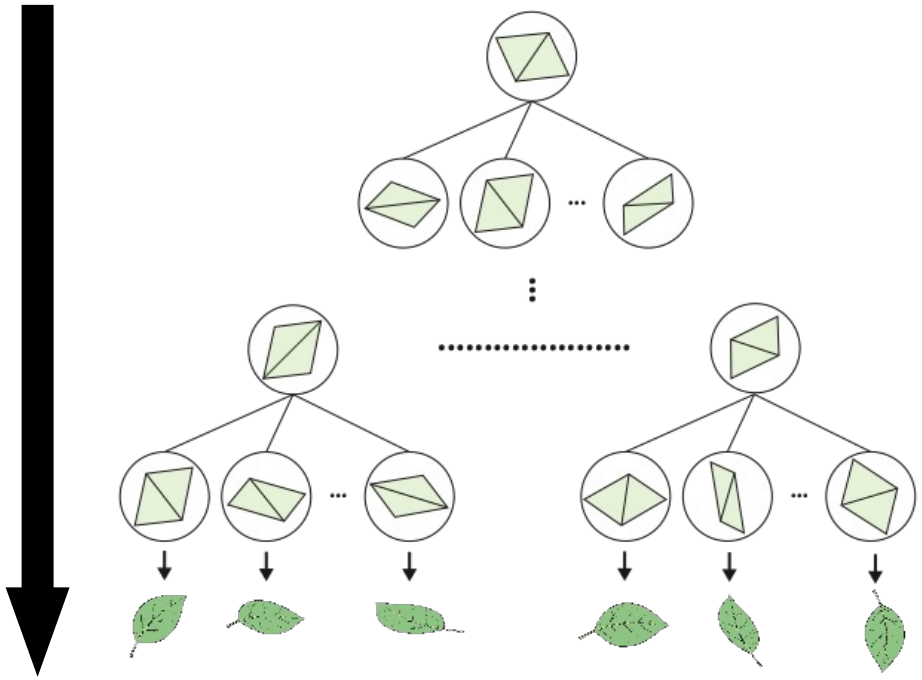
LOD 0
LOD 1
LOD 2

Pipeline Design Summary

- Compress the data based on the feature of tree leaves.
- Design leaf-level granularity LODs and organize them hieratically.
- Use GPU driven rendering pipeline (mesh shader)
  - Reduce I/O cost between CPU and GPU.
  - Avoid writing a buffer before rendering.

Culling (1)

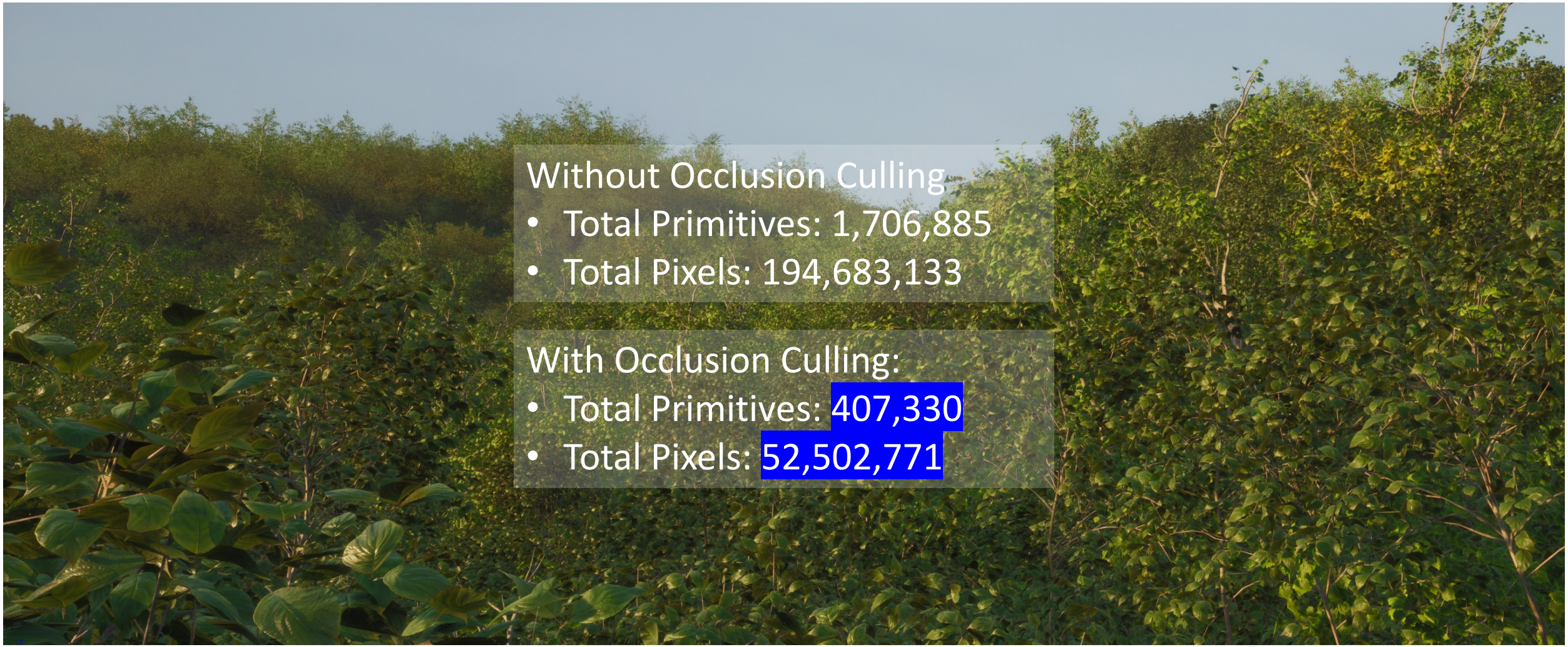
- Frustum culling and occlusion culling
  - Level by level according to the LOD tree.
  - Granularity: quad-leaf
  - Can handle layer-on-layer leaves.



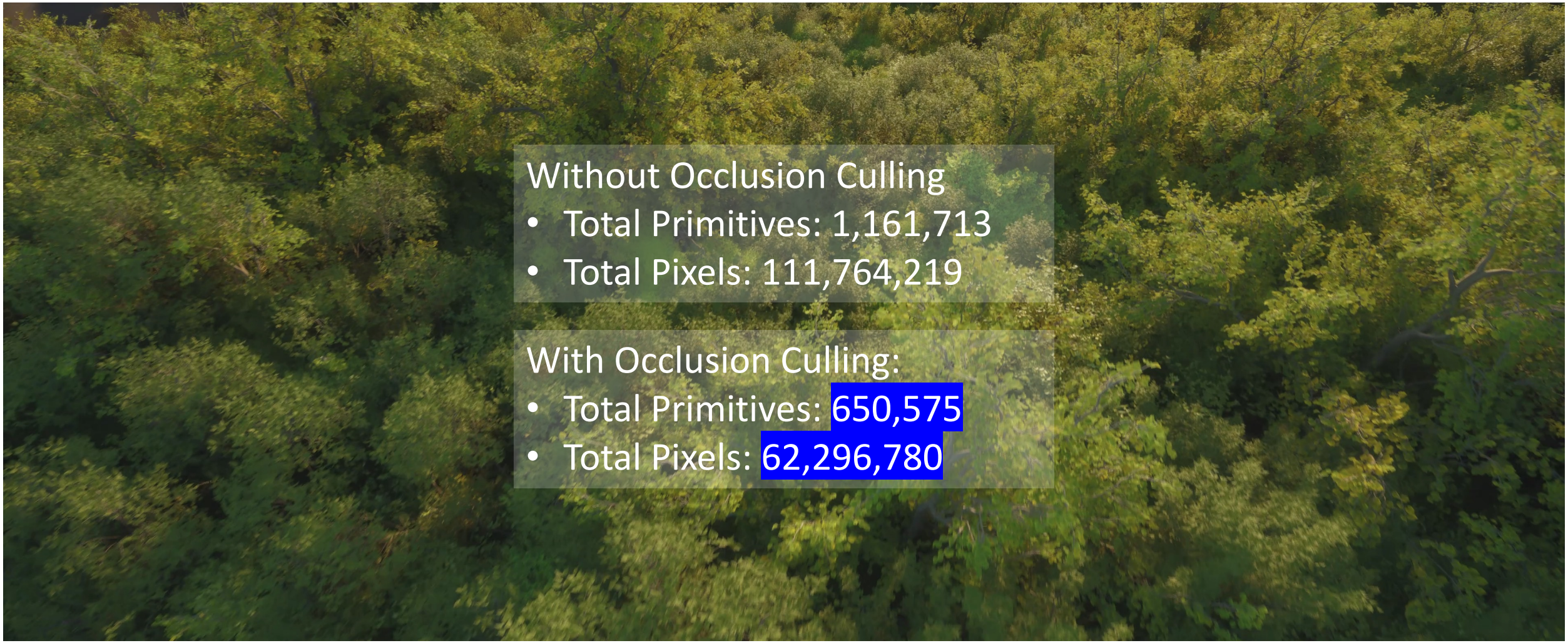
Culling (2)

- Hard to apply occlusion culling on foliage structure:
  1. Depth pre-pass with alpha test doubles the timing.
  2. Structures like foliage are hard to be well-occluded.
- Our method enables the access to the fine granularity, which increase the culling effectiveness.

Occlusion Culling Result (1)

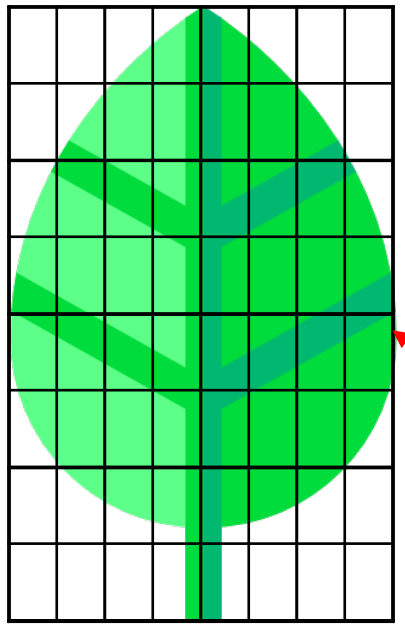


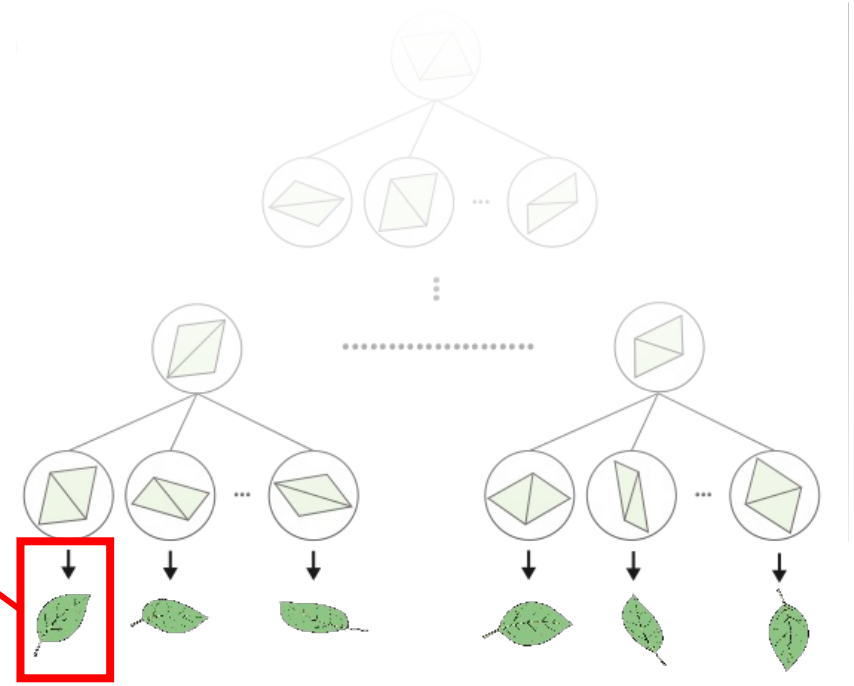
Occlusion Culling Result (2)



Subdivision

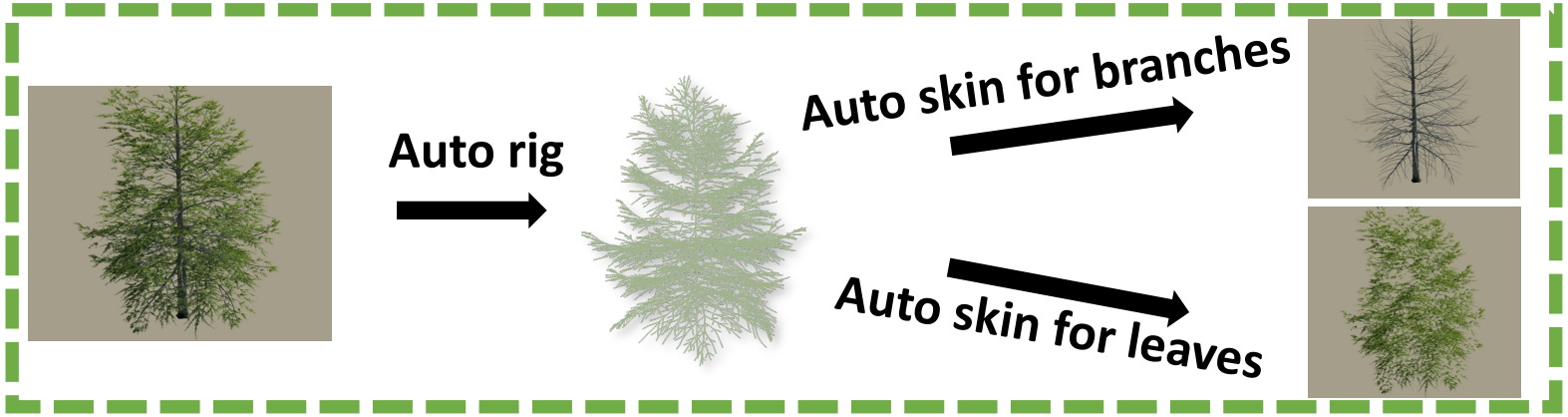
- Subdivision for coarse input models.
- Subdivide the LOD level-0 quad-leaves.
- Curling after subdivision.





Foliage Interaction

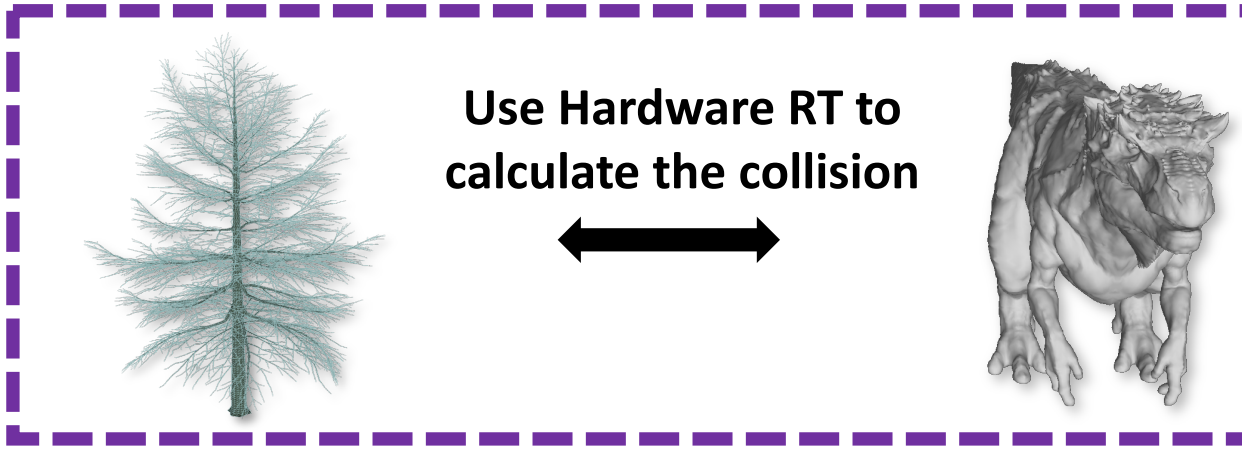
Preprocess



Runtime simulation

Tree Simulation

Update the tree BVH for Collision

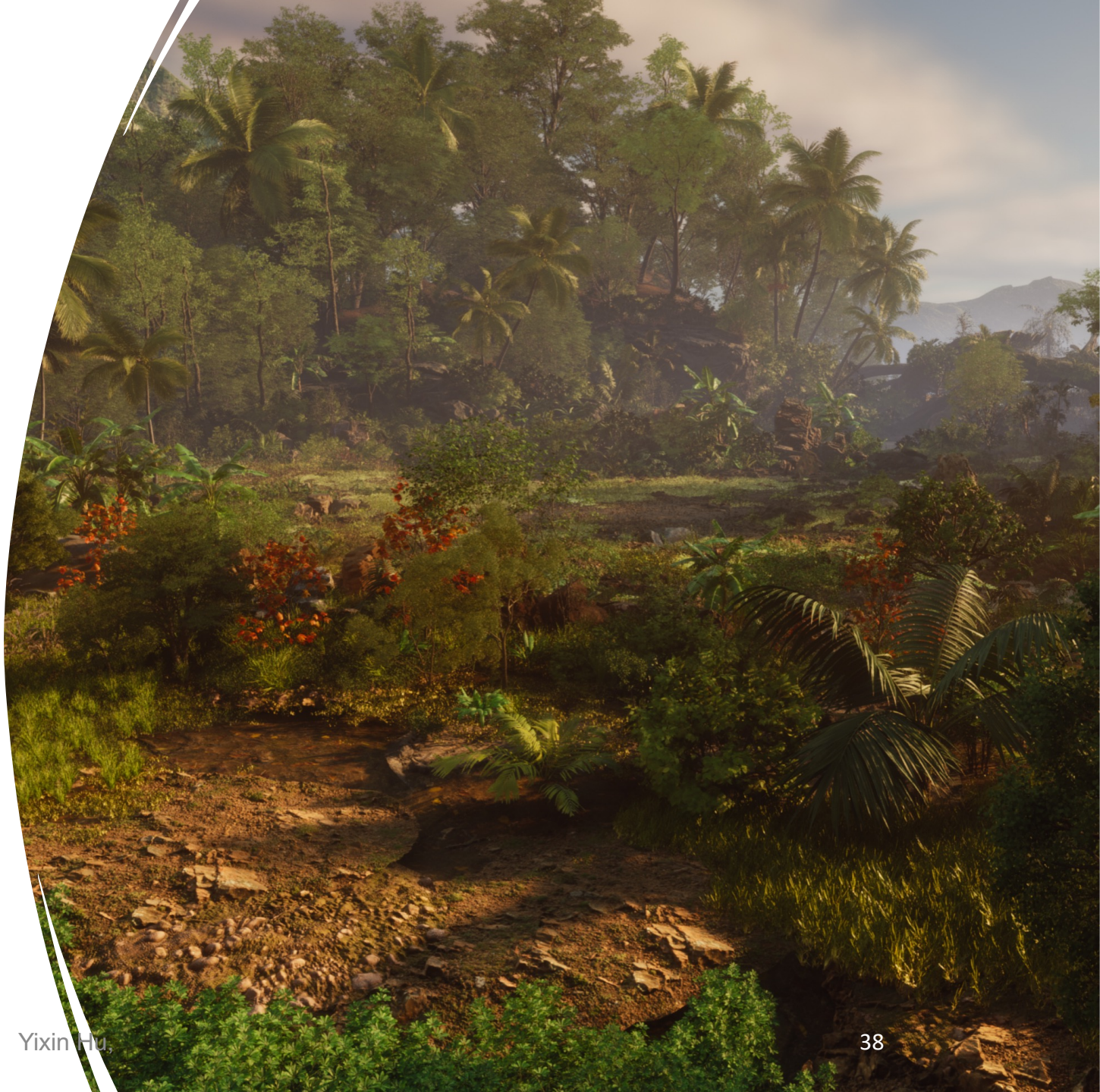


Generate rays from every mesh point; Use vertex normal as the ray direction.

Foliage Interaction Result

PART III: Comparison

Comparison with Nanite Foliage



Comparison (1): Efficiency

Ours: 3~4ms

START Pixel Lab

Comparison (2): Efficiency

Nanite foliage: ~37ms

Ours: 3~4ms

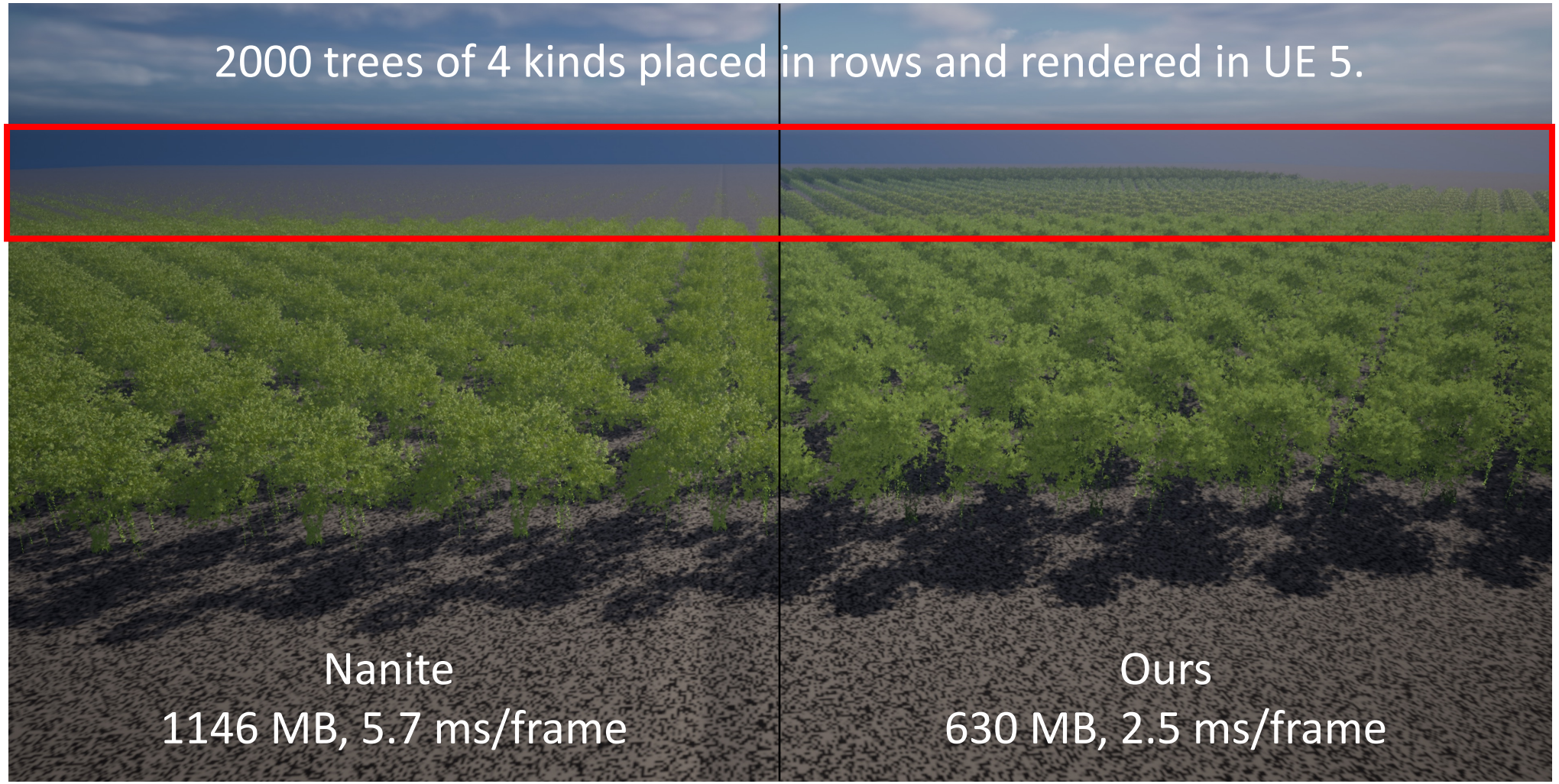
START Pixel Lab

Comparison (3): Efficiency

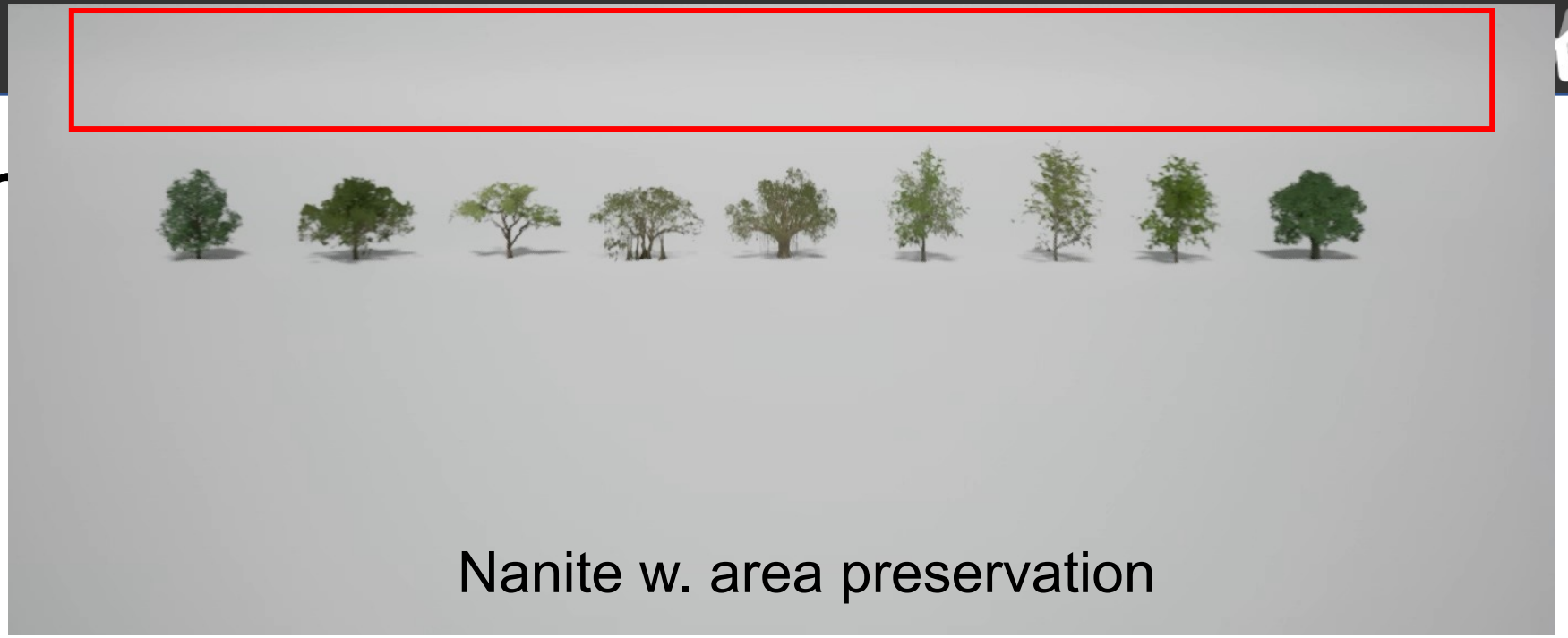
Ours: 3~4ms

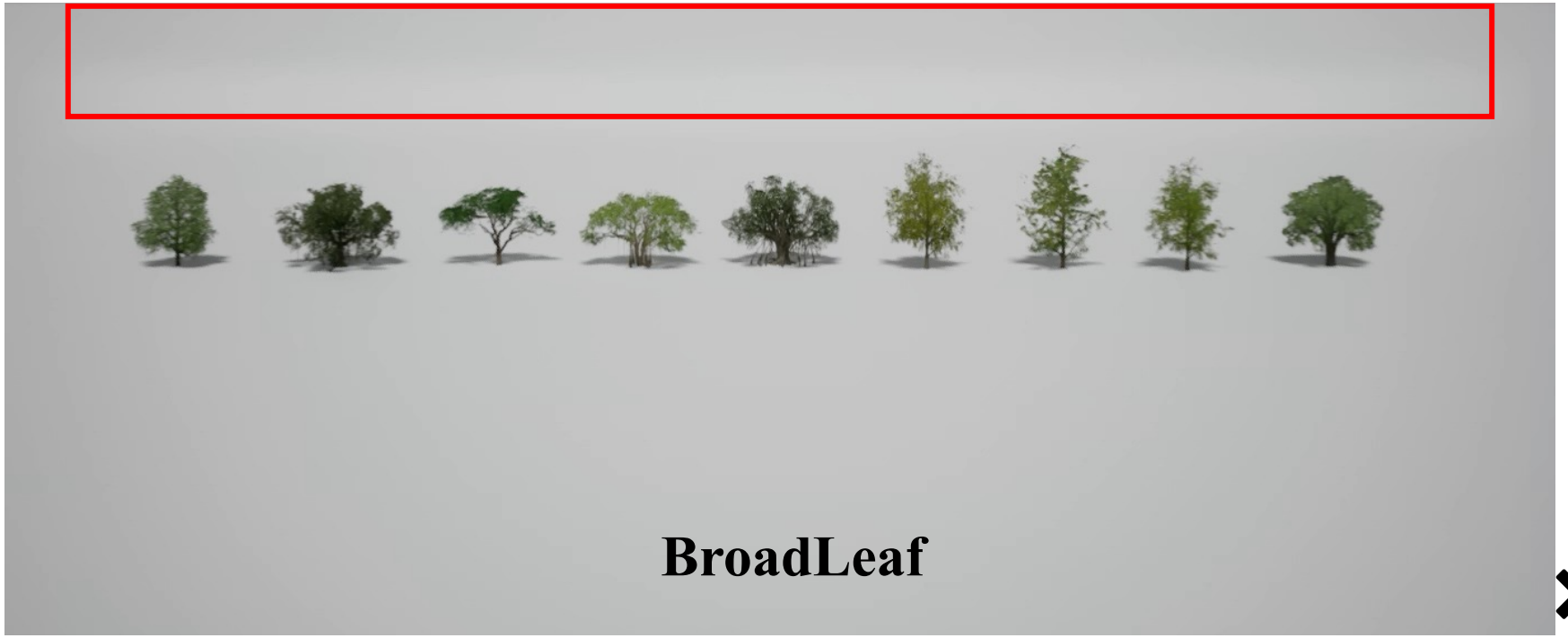
START Pixel Lab

Comparison (2): Faithfulness



Com

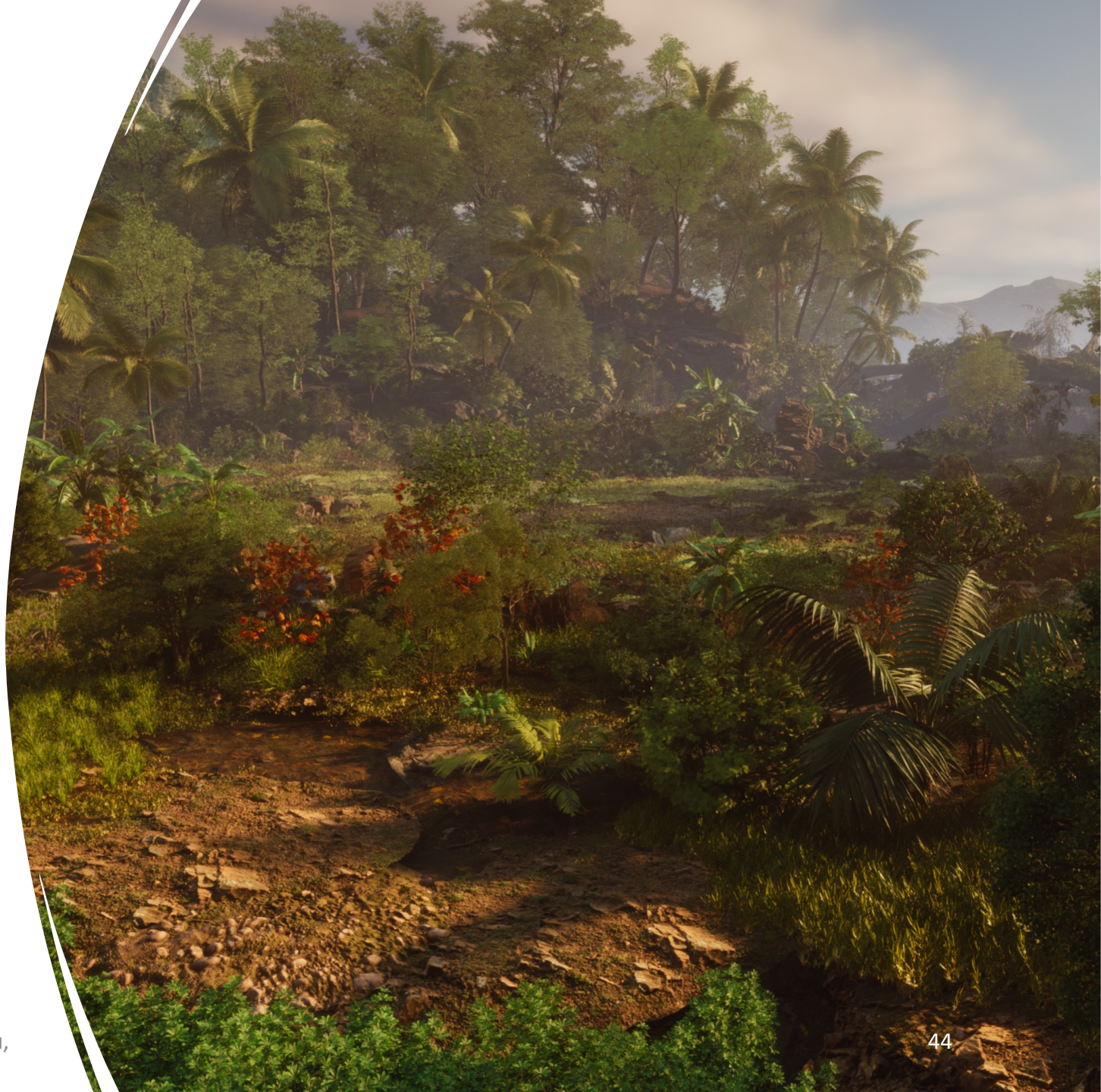




PART IV: Conclusion

Final Demo

Limitation & Future Work



START ENGINE

腾讯原生云游戏技术解决方案

Application

- Video gaming
- Virtual reality and augmented reality
- Geographic environment visualization
- Filmmaking (e.g. virtual production)

Limitation & Future Work (1)

- Better LOD mesh generation especially for the higher levels (low-poly meshes)
  - Smoother transition with even fewer elements.

Limitation & Future Work (2)

- BroadLeaf handles well the plants with leaf reference structure.
  - Non-leaf-reference structure: grass, banana tree, …
- Ray tracing-based method:
  - No need to generate LODs.
  - Shadow: more efficient and more stable.
  - The cost increases slower when the complexity of the scene increases.

End

Thank you!