Lecture 06

Rendering on Game Engine

The Challenges and Fun of Rendering the Beautiful Mother Nature

WANG XI

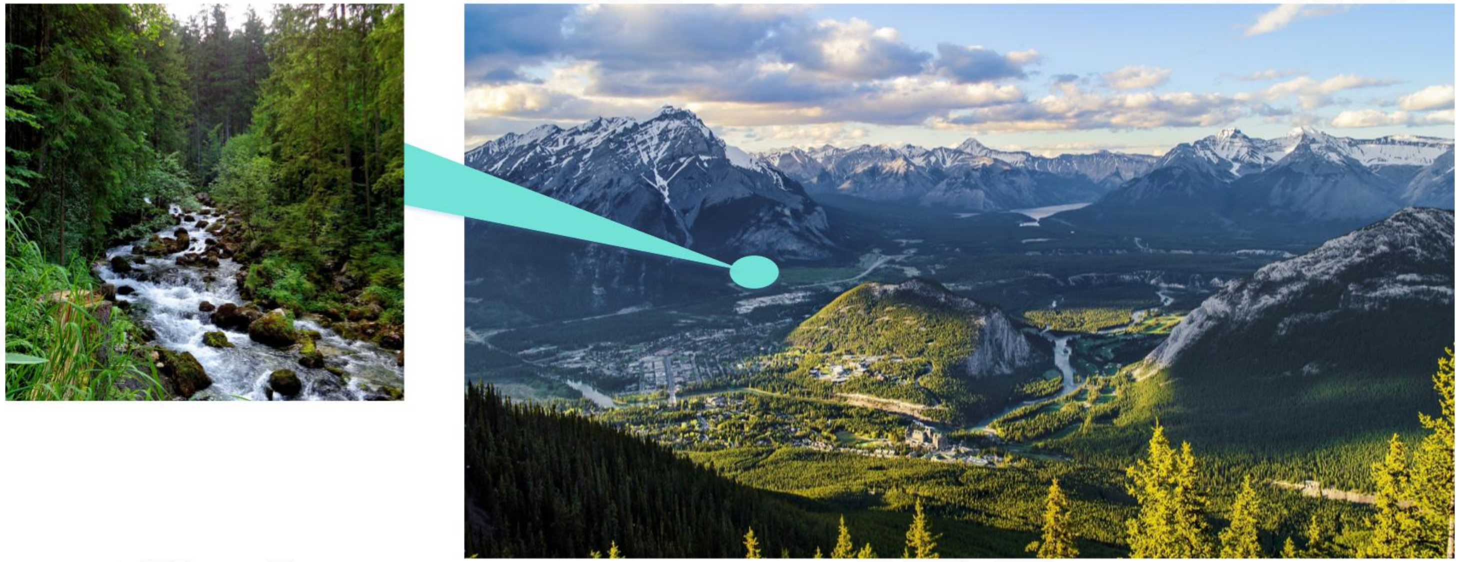
GAMES 104

2022


Red Dead Redemption

Real-World Landscape

- Huge geospatial scale
- Rich geomorphological
    - Vegetation
    - Rivers
    - Undulating peaks
    - Alpine snow
    - ……



Too Complex for Rendering Using Traditional Mesh + Material

Environment Components in Games

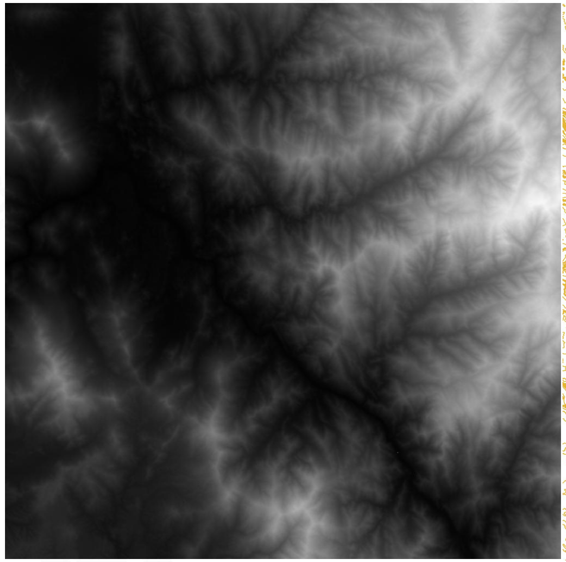
Terrain Rendering


Microsoft Flight Simulator

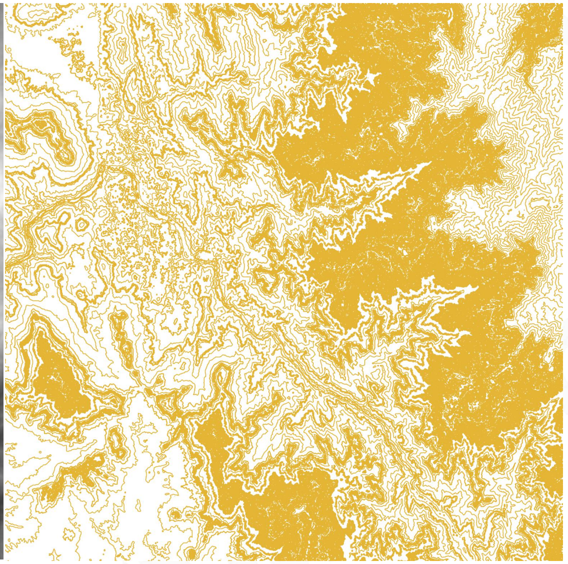

No Man’s Sky

Simple Idea - Heightfield

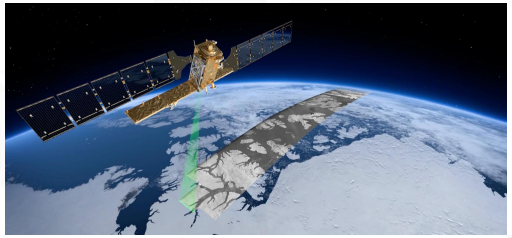
- Satellite image and google earth

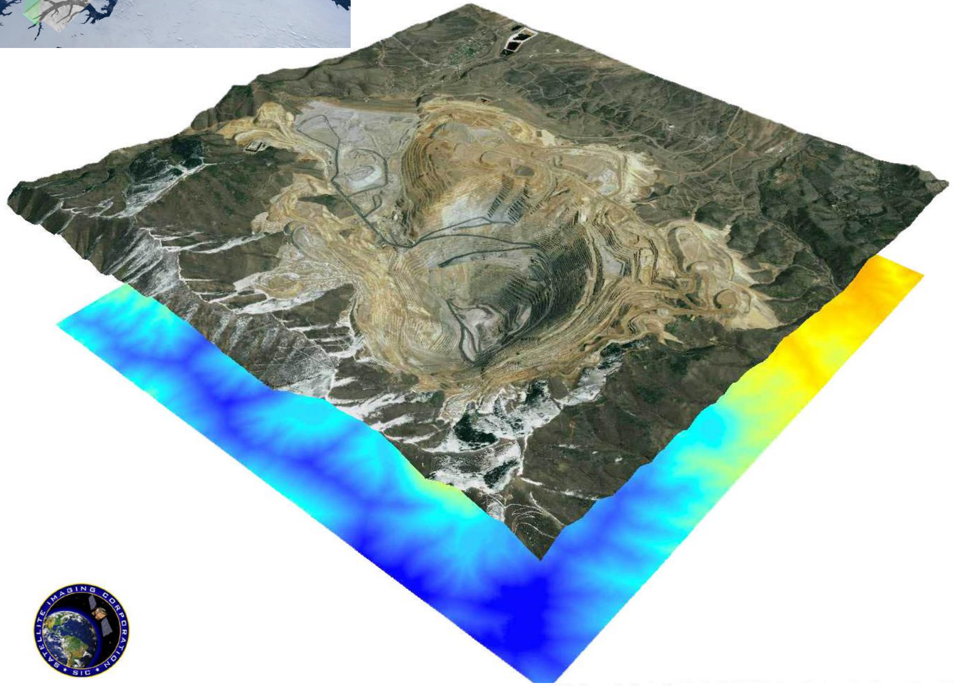


Height Map



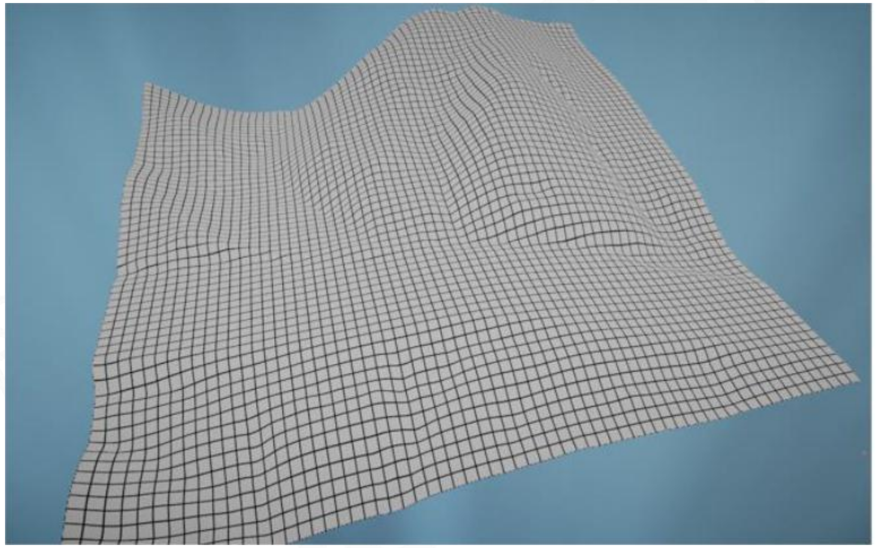
Contour Map





Expressive Heightfield Terrains

Render Terrain with Heightfield



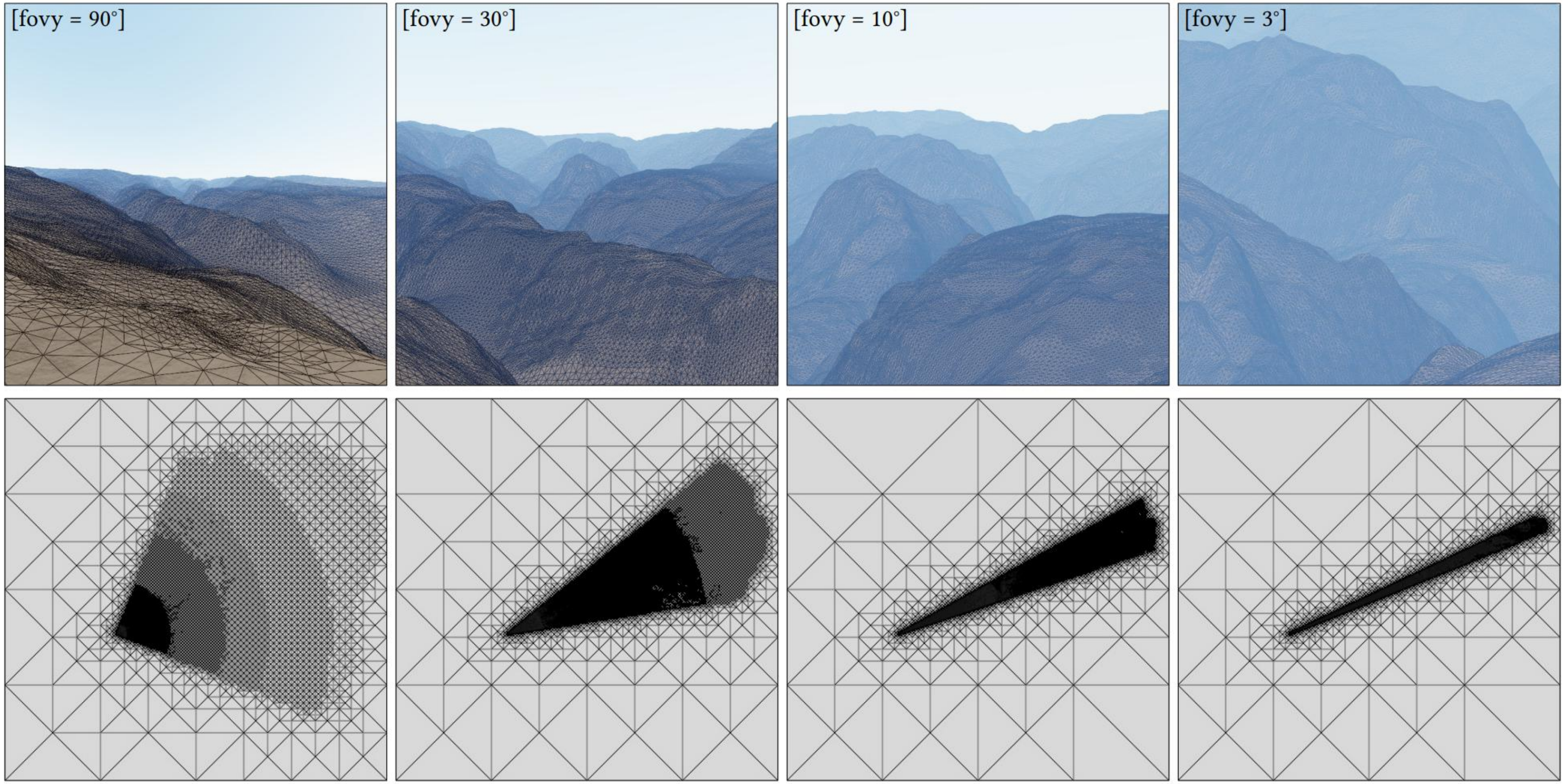

Mesh Grids


Material

1km × 1km map, sample distance 1m
Need 2 * 1,000 * 1,000 = 2,000,000 triangles


Vast Open World

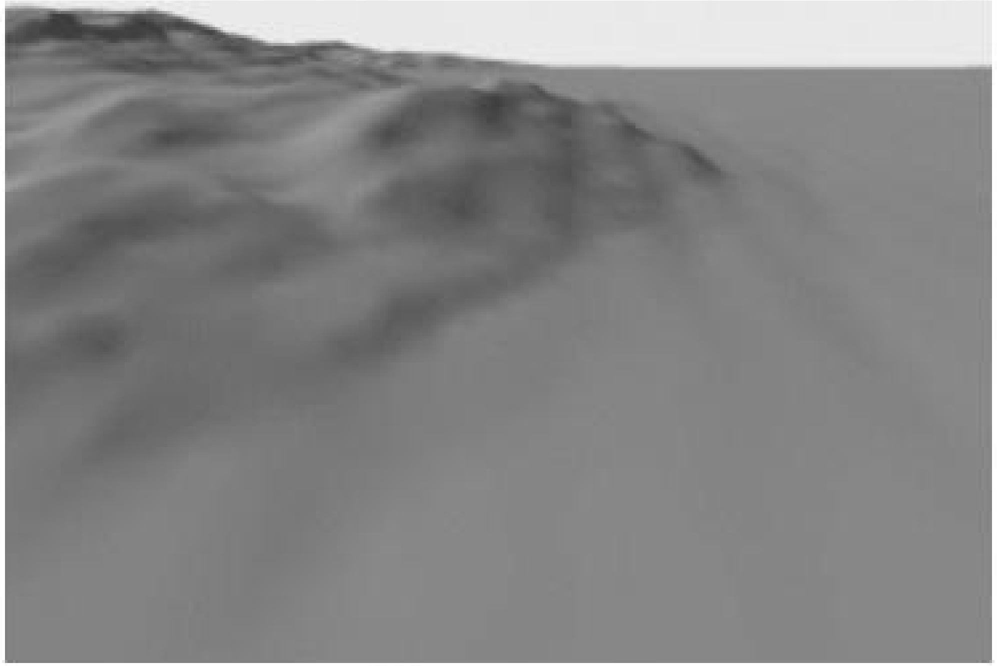
Adaptive Mesh Tessellation

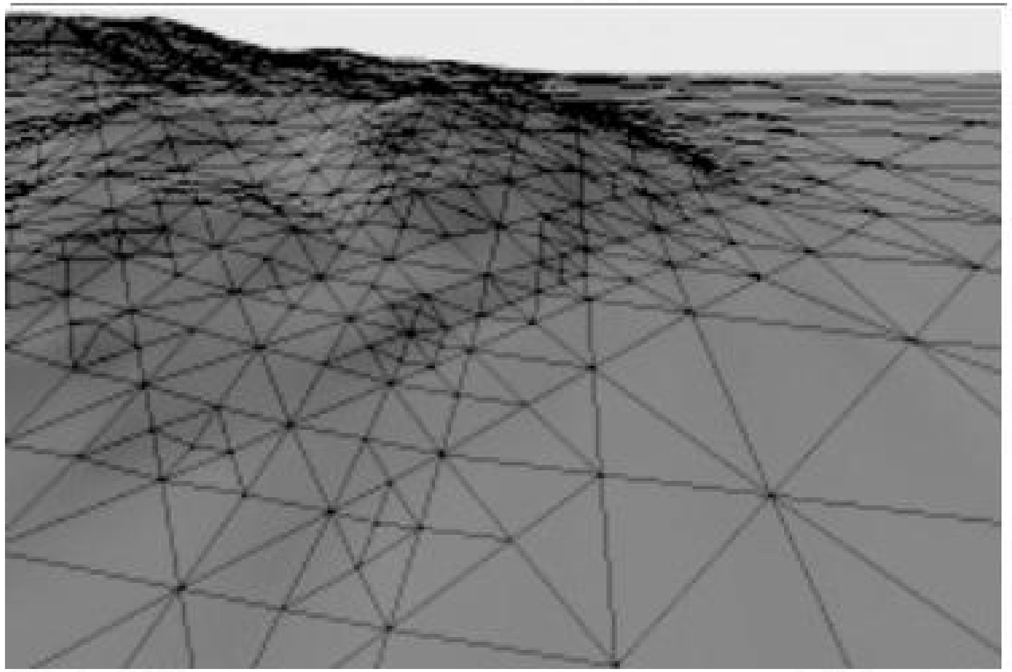


Two Golden Rules of Optimization

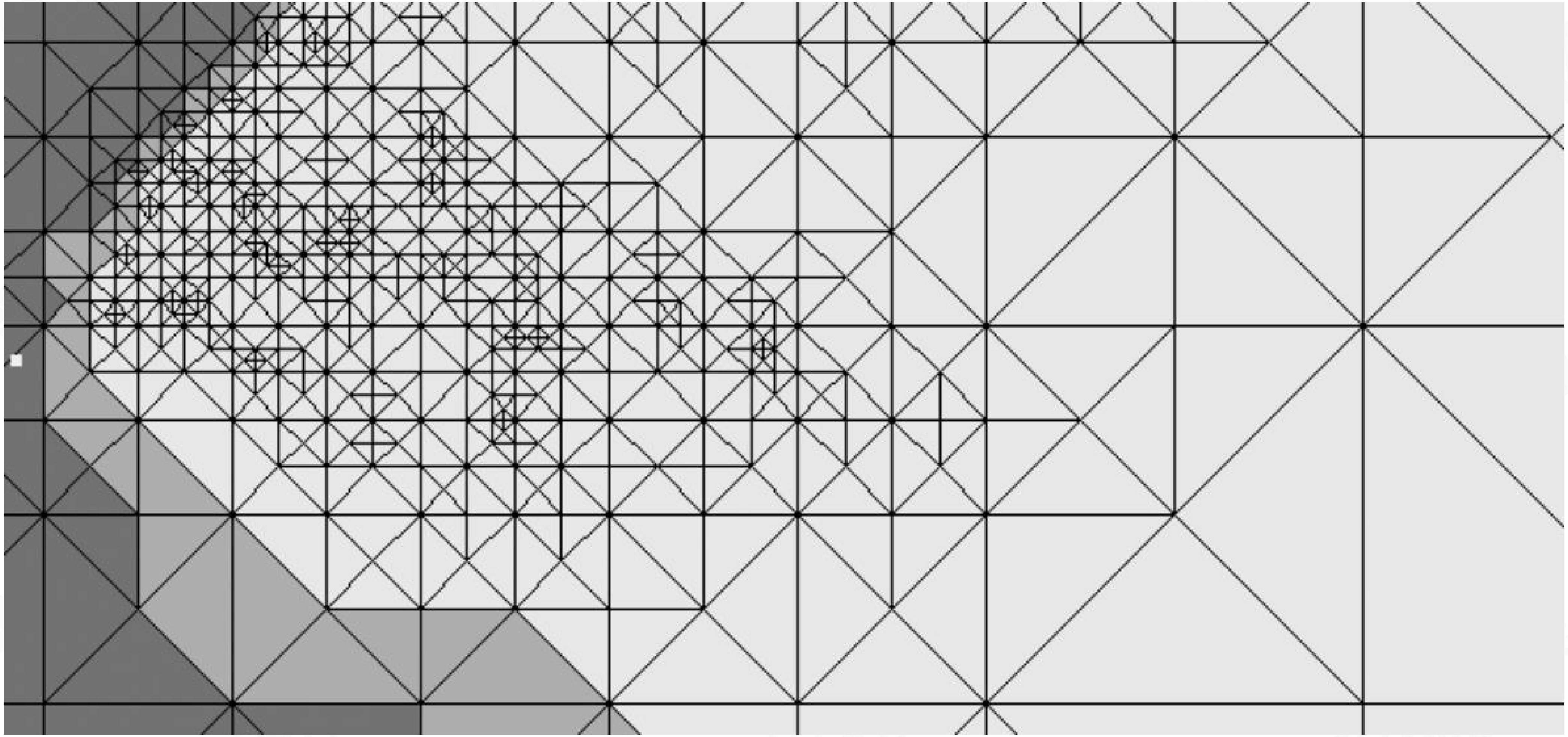
View-dependent error bound

- Distance to camera and FoV
- Error compare to ground truth (pre-computation)



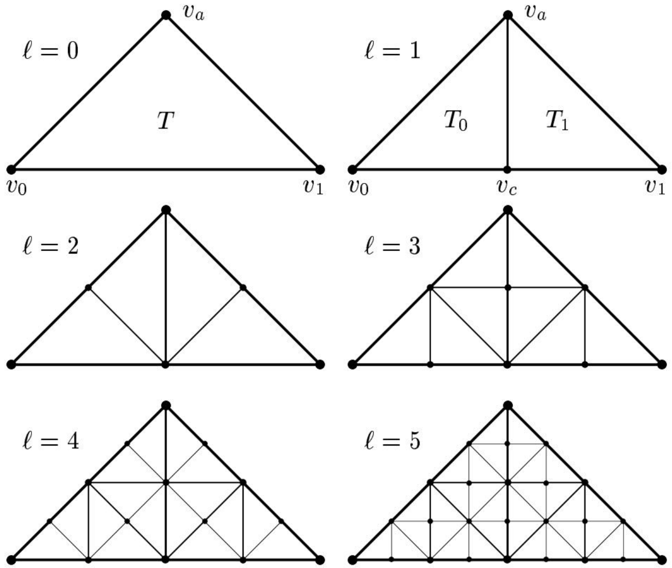


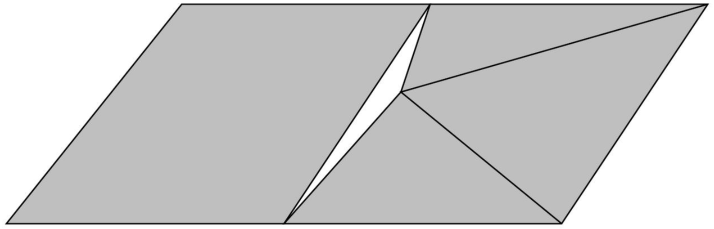
Triangle-Based Subdivision



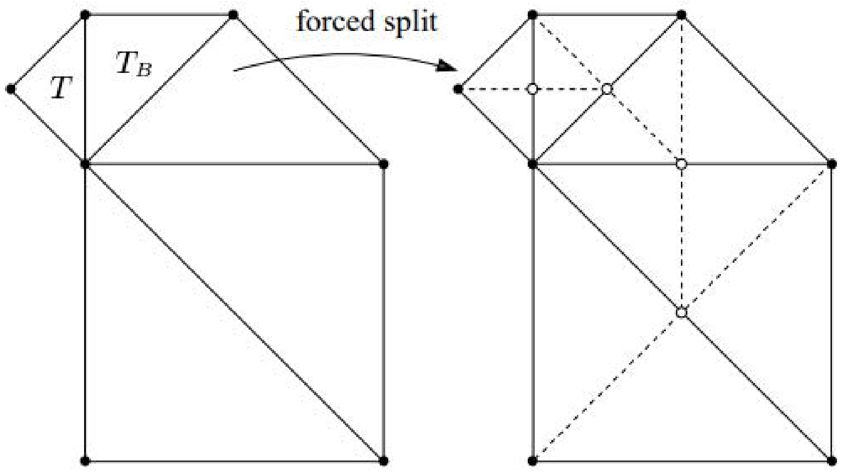
Subdivision and T-Junctions

Continuously partitioning triangles and their children based on the idea of binary trees





T-Junction



Triangle-Based Subdivision on GPU

54×54 km terrain on GPU using Unity game engine

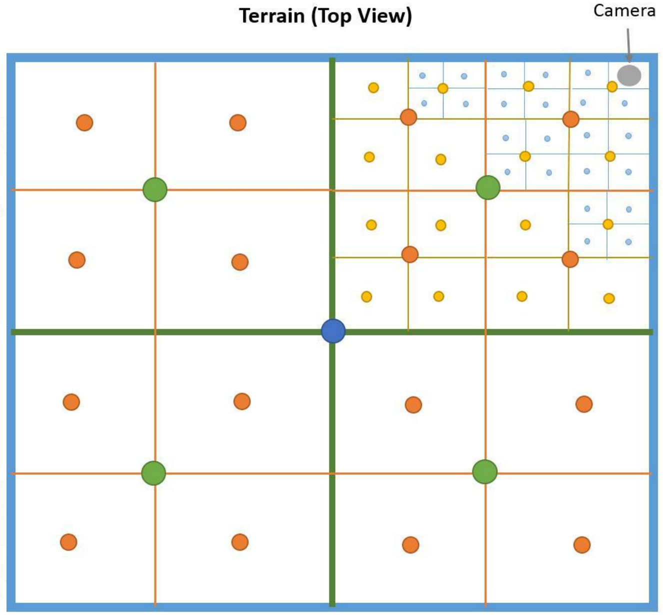
QuadTree-Based Subdivision

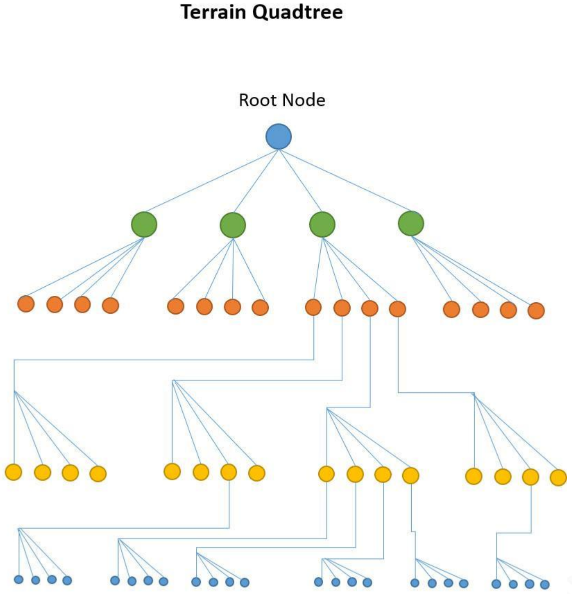
Pros

- Easy to construct
- Easy management of data under geospatial, including objects culling and data streaming

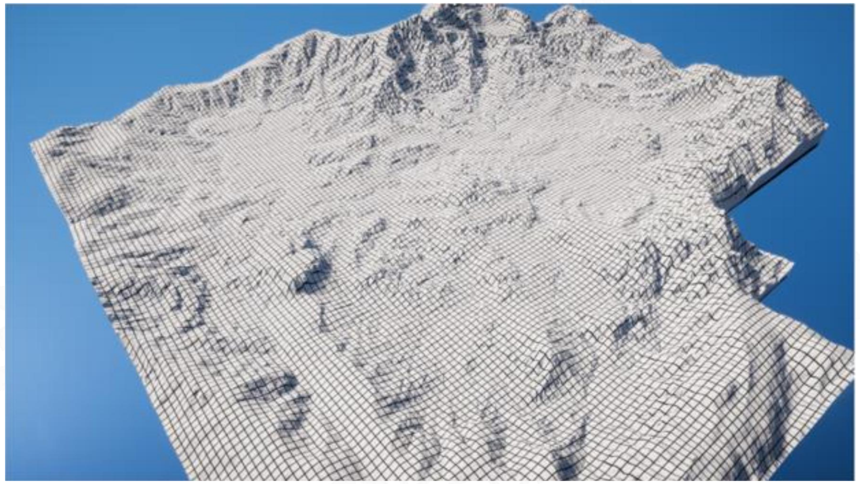
Cons

- Mesh subdivision is not as flexible as triangle mesh
- The grid level of the leaf nodes needs to be consistent



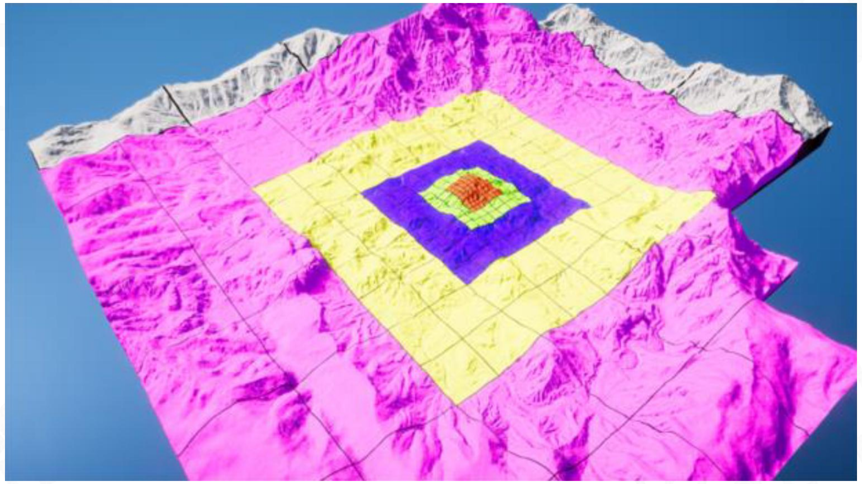


Original Terrain



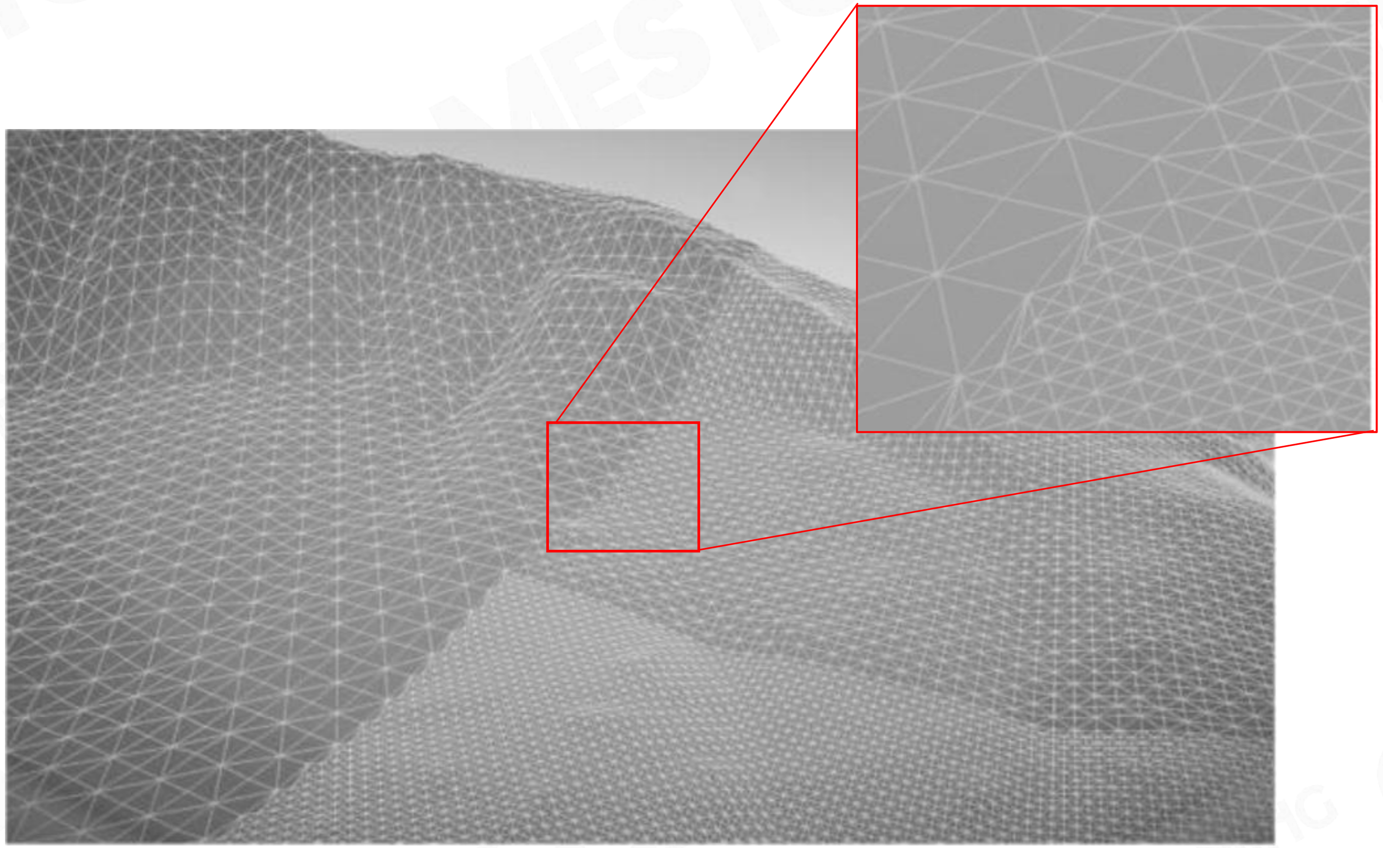
Highest Resolution Grid

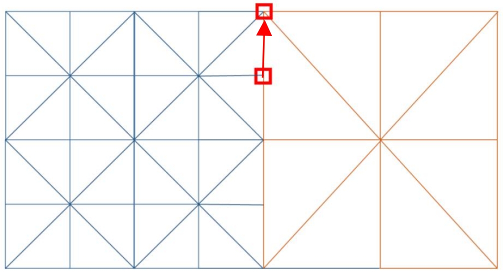
Lowest Resolution Grid



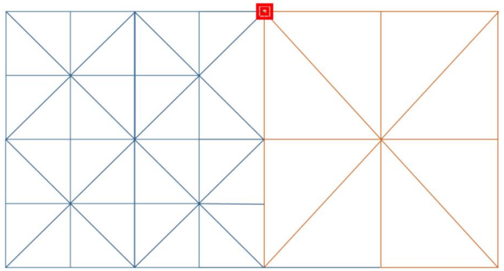
Terrain Quad Tree

Solving T-Junctions among Quad Grids

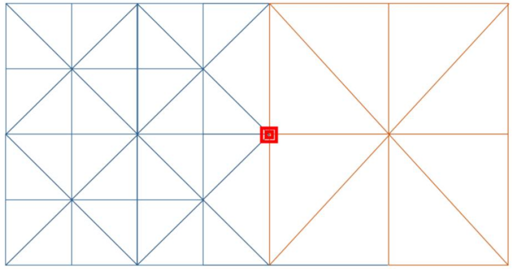




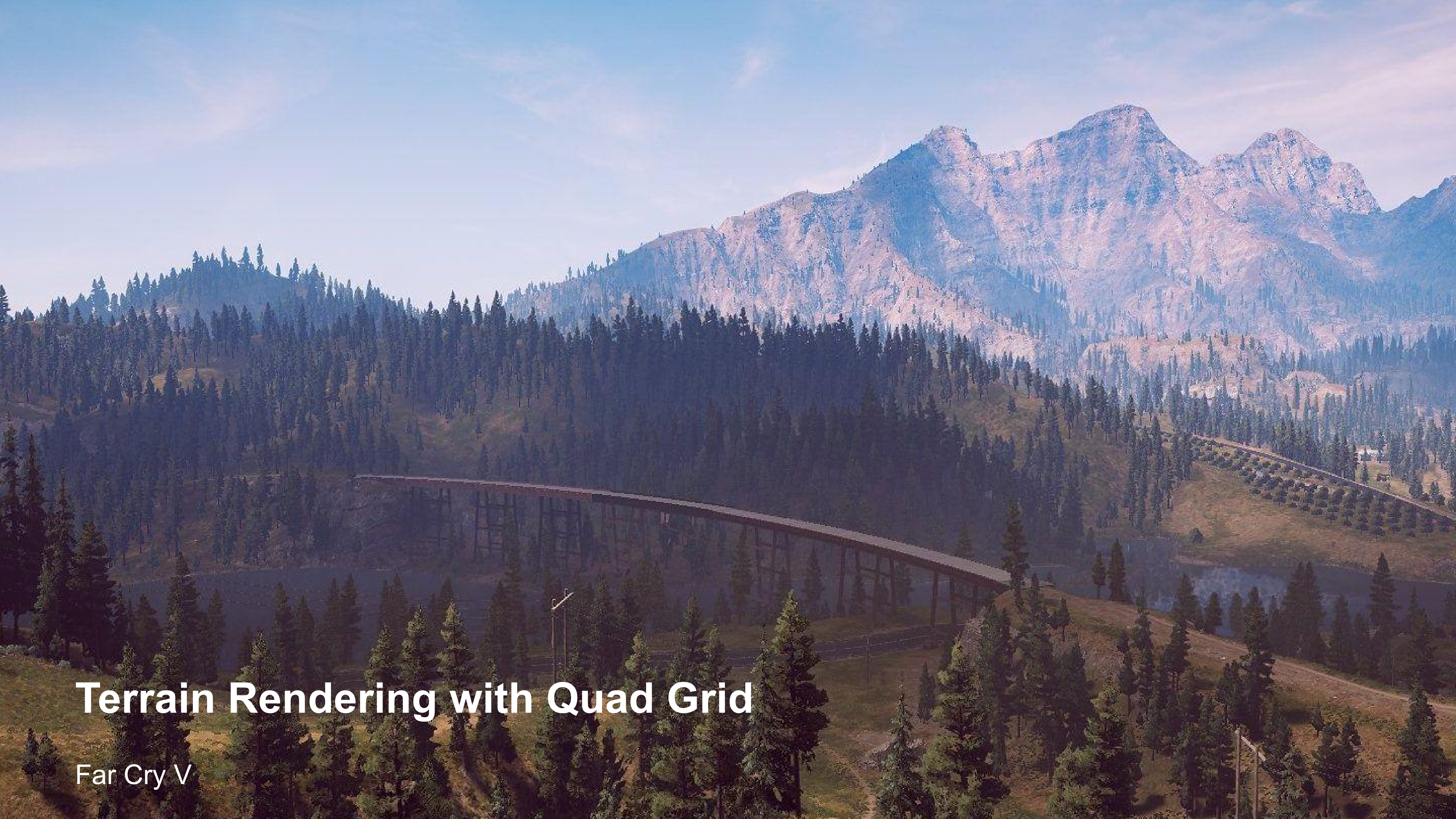
Source



Stitching Step 1



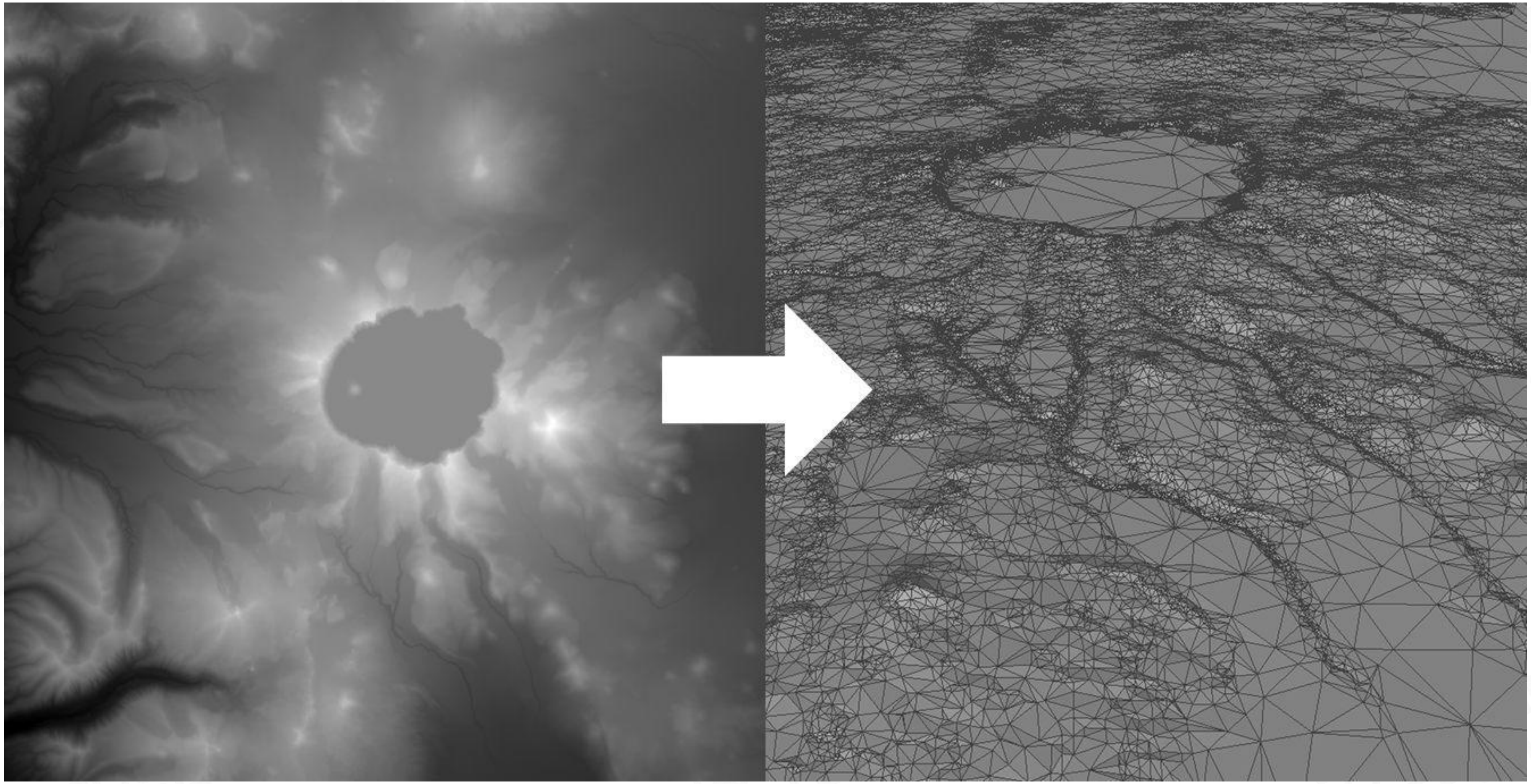
Mesh LoD Stitching

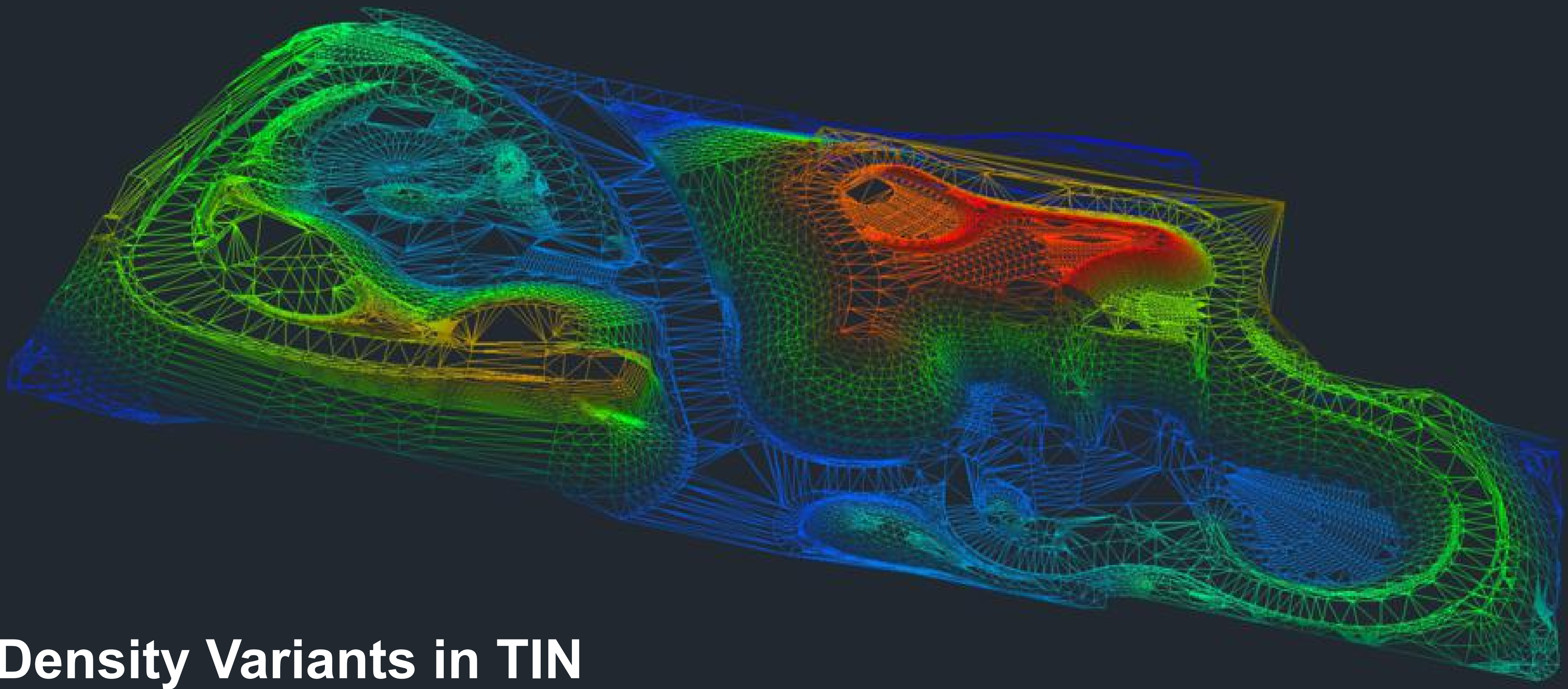


Terrain Rendering with Quad Grid

Far Cry V

Triangulated Irregular Network (TIN)





Density Variants in TIN

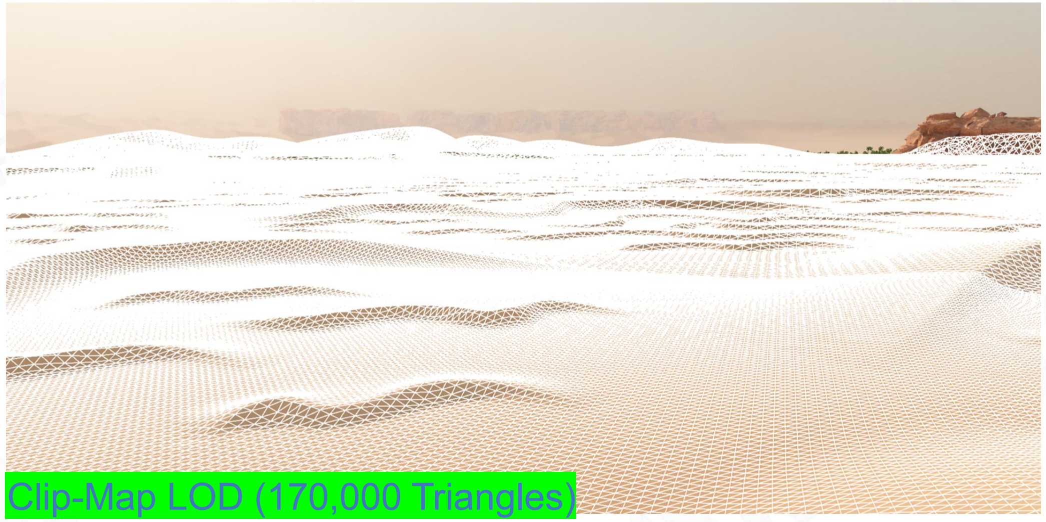
Triangulated Irregular Network vs. Adpative Tessellation

Pros

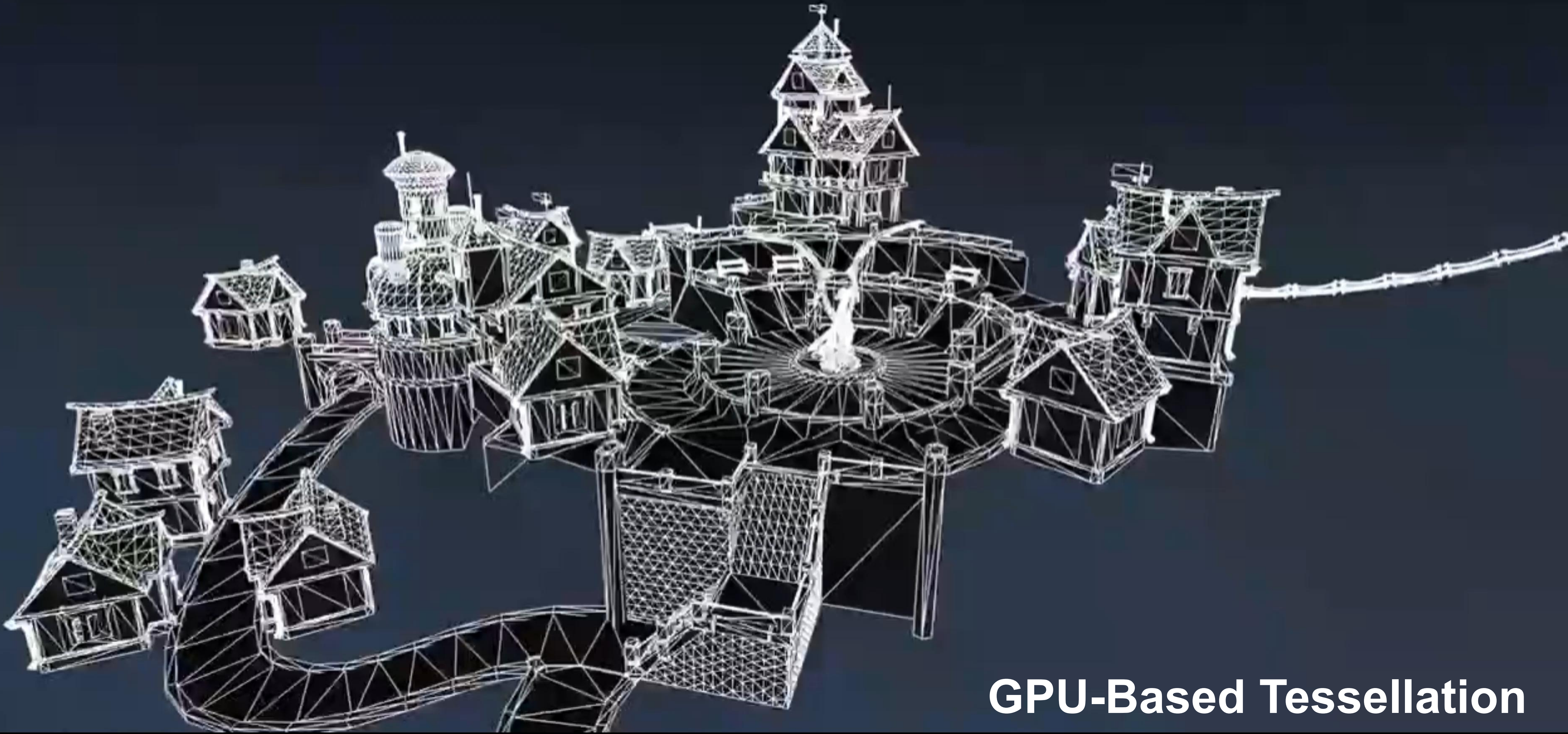
- Easy in runtime rendeirng
- Less triangls in certain terrain types

Cons

- Requires certain pre-processing steps
- Poor reusability

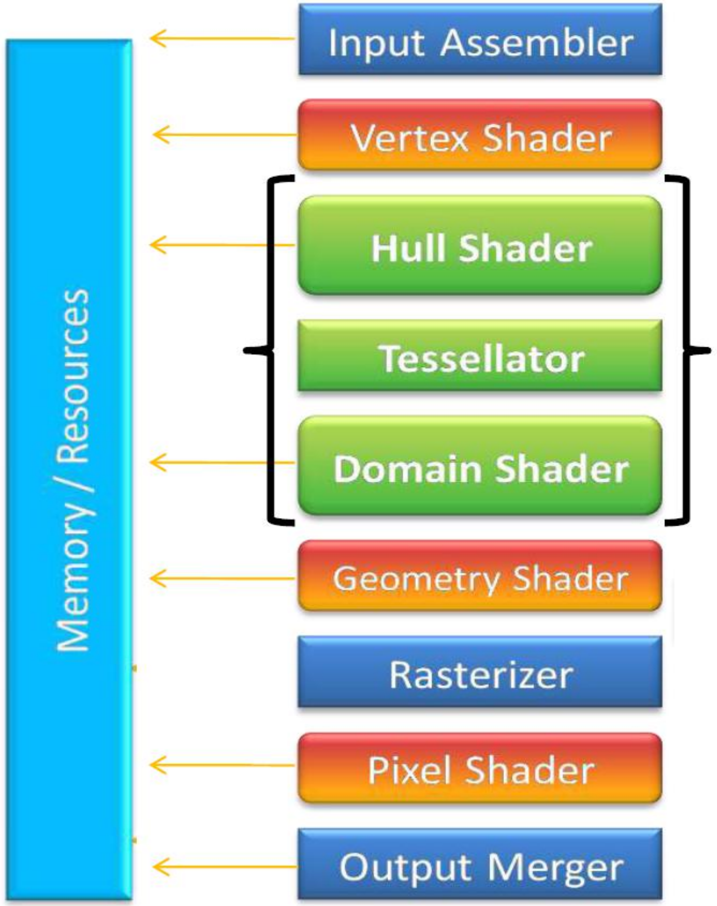


GDC2021 Boots on the Ground: The Terrain of Call of Duty



GPU-Based Tessellation

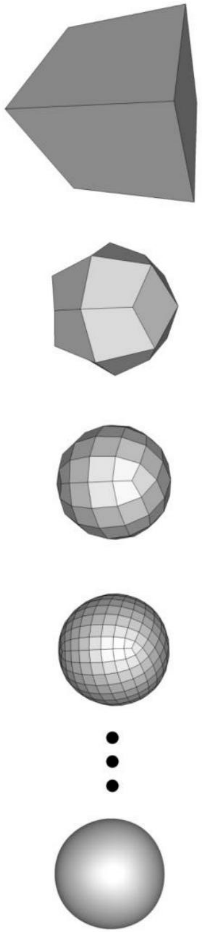
Hardware Tessellation



Hull-Shader Stage - transforms basis functions from base mesh to surface patches

Tessellator Stage - produces a semi-regular tessellation pattern for each patch

Domain-Shader Stage - a programmable shader stage that calculates the vertex position that corresponds to each domain sample



Simple Hull Shader (control point phase)

```
// Called once per control point
[domain("tri")] // indicates a triangle patch (3 verts)
[partitioning("fractional_odd")] // fractional avoids popping
// vertex ordering for the output triangles
[outputtopology("triangle_cw")] [outputcontrolpoints(3)]
// name of the patch constant hull shader
[patchconstantfunc("ConstantsHS")]
[maxtessfactor(7.0)] //hint to the driver – the lower the better
// Pass in the input patch and an index for the control point
HS_CONTROL_POINT_OUTPUT HS( InputPatch<VS_OUTPUT_HS_INPUT, 3>
inputPatch, uint uCPID : SV_OutputControlPointID )
{
    HS_CONTROL_POINT_OUTPUT Out;

    // Copy inputs to outputs – "pass through" shaders are optimal
    Out.vWorldPos = inputPatch[uCPID].vPosWS.xyz;
    Out.vTexCoord = inputPatch[uCPID].vTexCoord;
    Out.vNormal = inputPatch[uCPID].vNormal;
    Out.vLightTS = inputPatch[uCPID].vLightTS;

    return Out;
```

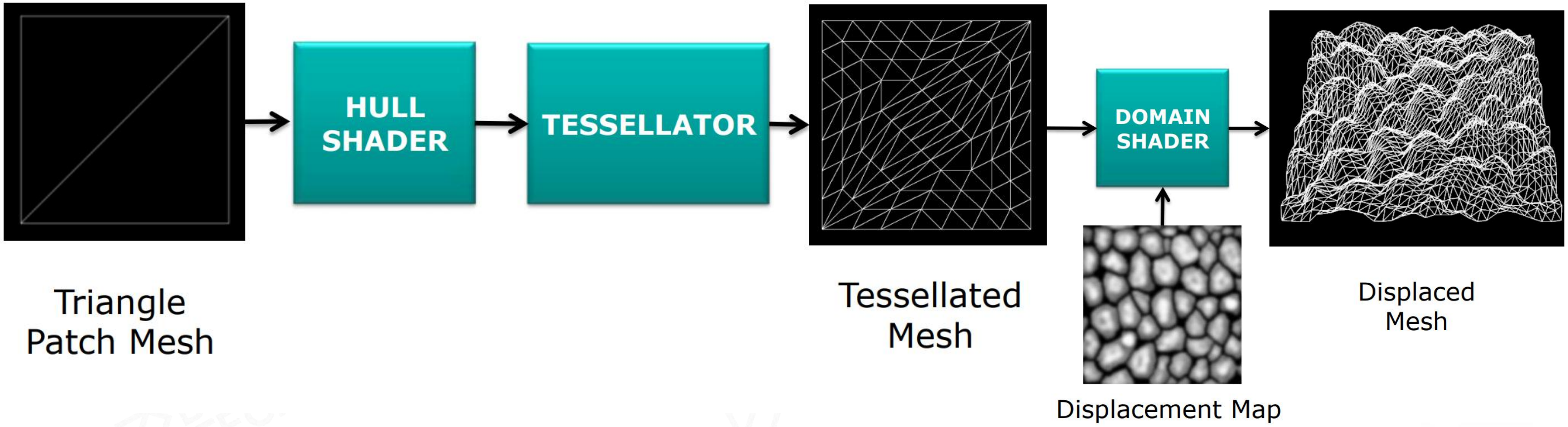
Simple Hull Shader (patch constant phase)

```
//Called once per patch. The patch and an index to the patch (patch
// ID) are passed in
HS_CONSTANT_DATA_OUTPUT ConstantsHS( InputPatch<VS_OUTPUT_HS_INPUT, 3>
   p, uint PatchID : SV_PrimitiveID )
{
    HS_CONSTANT_DATA_OUTPUT Out;

   // Assign tessellation factors – in this case use a global
   // tessellation factor for all edges and the inside. These are
   // constant for the whole mesh.
   Out.Edges[0] = g_TessellationFactor;
   Out.Edges[1] = g_TessellationFactor;
   Out.Edges[2] = g_TessellationFactor;
   Out.Inside   = g_TessellationFactor;
   return Out;
}
```

Simple Domain Shader (part 1)

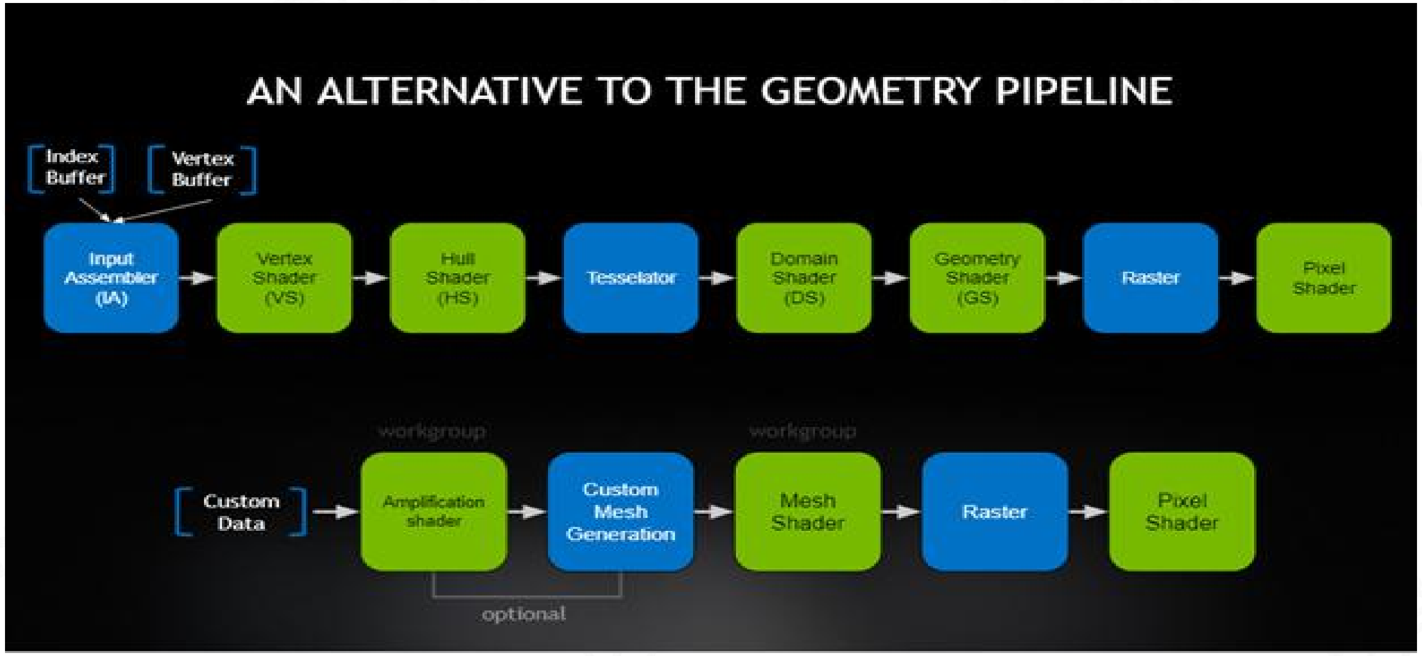
```
// Called once per tessellated vertex
[domain("tri")] // indicates that triangle patches were used
// The original patch is passed in, along with the vertex position in barycentric
coordinates, and the patch constant phase hull shader output (tessellation factors)
DS_VS_OUTPUT_PS_INPUT DS( HS_CONSTANT_DATA_OUTPUT input,
             float3 BarycentricCoordinates : SV_DomainLocation,
             const OutputPatch<HS_CONTROL_POINT_OUTPUT, 3>
             TrianglePatch )
{
    DS_VS_OUTPUT_PS_INPUT Out;
    // Interpolate world space position with barycentric coordinates
    float3 vWorldPos =
        BarycentricCoordinates.x * TrianglePatch[0].vWorldPos +
        BarycentricCoordinates.y * TrianglePatch[1].vWorldPos +
        BarycentricCoordinates.z * TrianglePatch[2].vWorldPos;
    // Interpolate texture coordinates with barycentric coordinates
    Out.vTexCoord =
       BarycentricCoordinates.x * TrianglePatch[0].vTexCoord + ...
    // Interpolate normal with barycentric coordinates
    float3 vNormal =
       BarycentricCoordinates.x * TrianglePatch[0].vNormal + ...
```

```
    // sample the displacement map for the magnitude of displacement
    float fDisplacement = g_DisplacementMap.SampleLevel(
                                 g_sampleLinear, Out.vTexCoord.xy, 0 ).r;
    fDisplacement *= g_Scale;
    fDisplacement += g_Bias;
    float3 vDirection = -vNormal; // direction is opposite normal

    // translate the position
    vWorldPos += vDirection * fDisplacement;

    // transform to clip space
    Out.vPosCS = mul( float4( vWorldPos.xyz, 1.0 ),
                                 g_mWorldViewProjection );

    return Out;
} // end of domain shader
```

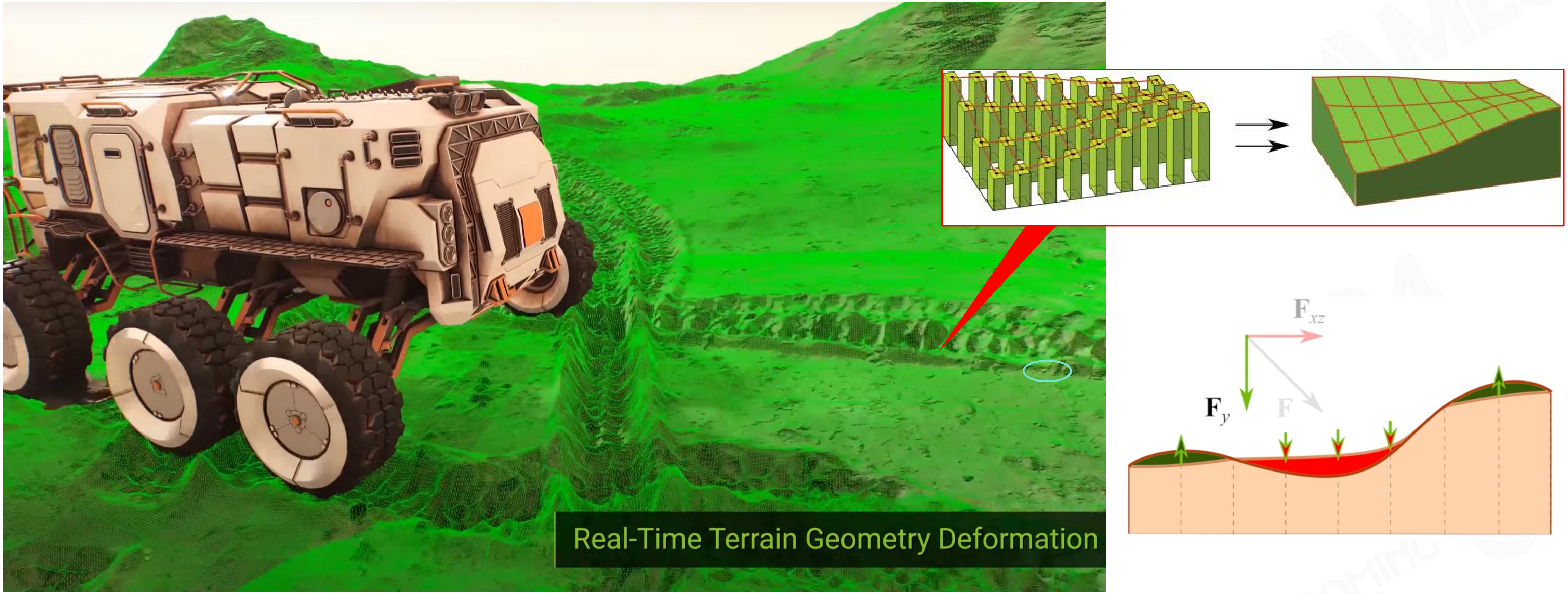


Mesh Shader Pipeline

- **Amplification Shader Stage** - decides how many Mesh shader groups to run and passes data to those groups
- **Mesh Shader Stage** - produces a semi-regular tessellation pattern for each patch, and outputs comprise vertices and primitives

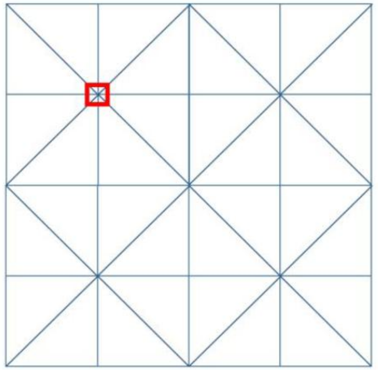


Real-Time Deformable Terrain

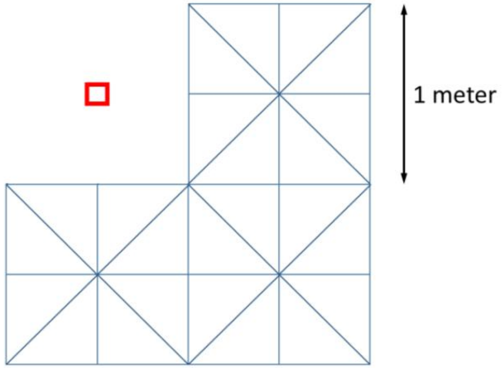



Non-Heightfield Terrain

Dig a Hole in Terrain



Output 1 NaN
Vertex Position

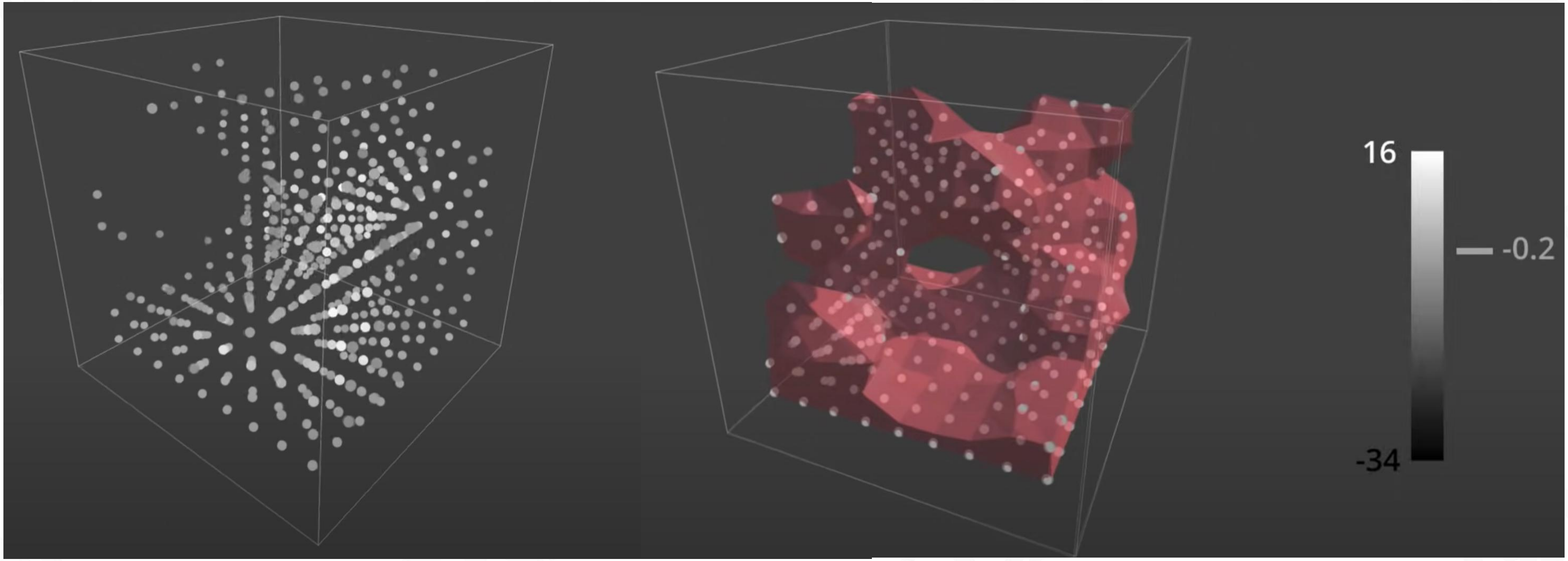


Kill a Quad

Cull a terrain vertex by outputting NaN from the vertex shader

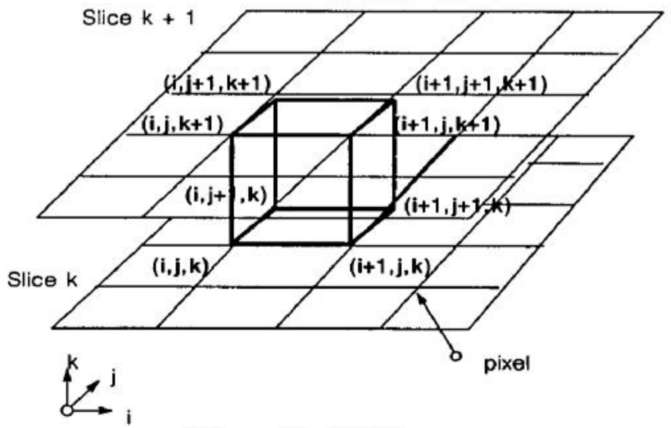
- `projectedPosition /= (isHole ? 0 : 1)`

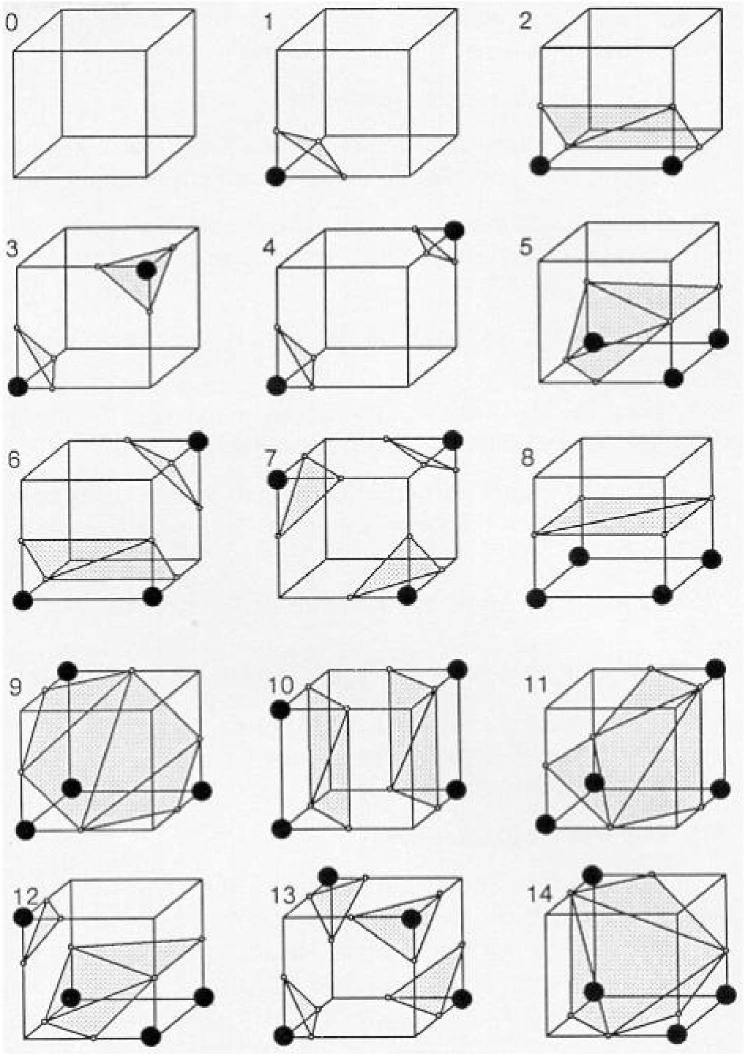
Crazy Idea - Volumetric Representation

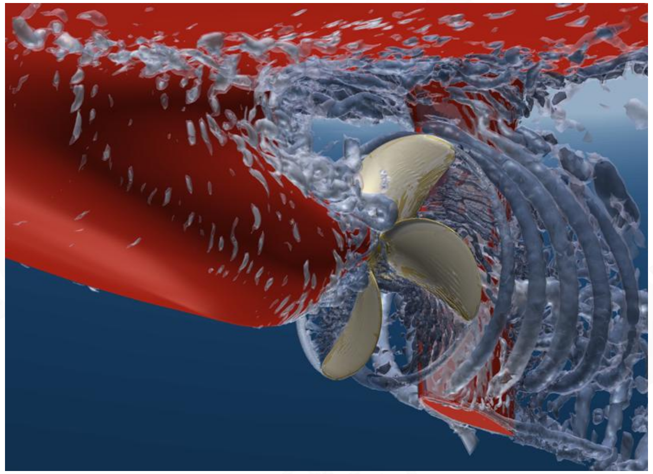


In 3D computer graphics, a voxel represents a value on a regular grid in three-dimensional space. As pixels in a 2D bitmap, voxels themselves do not typically have their position (i.e. coordinates) explicitly encoded with their values

Marching Cubes





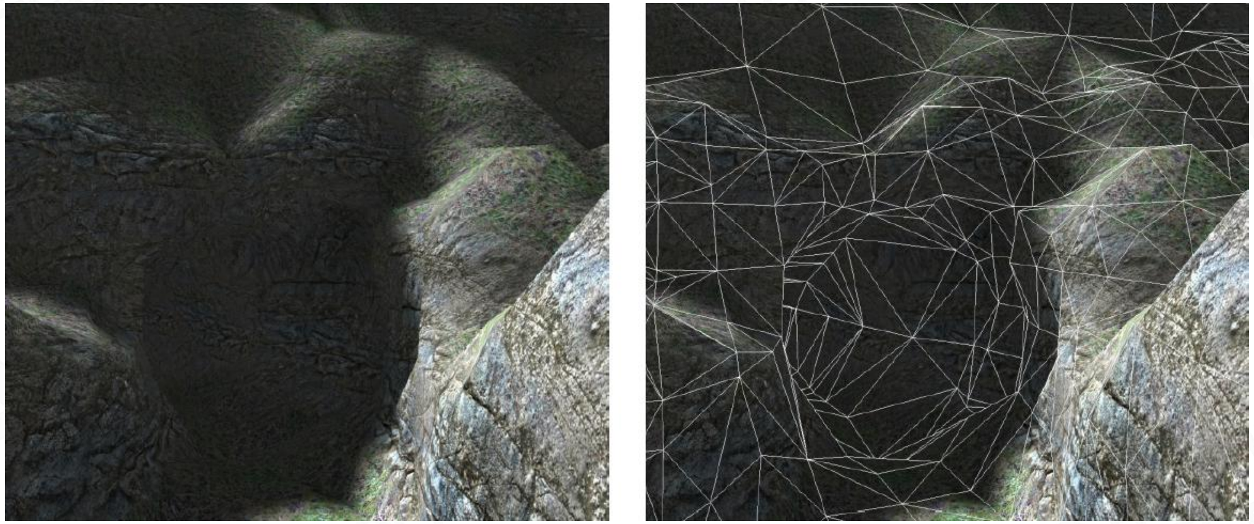


MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM'; Computer Graphics, Volume 21, Number 4, July 1987

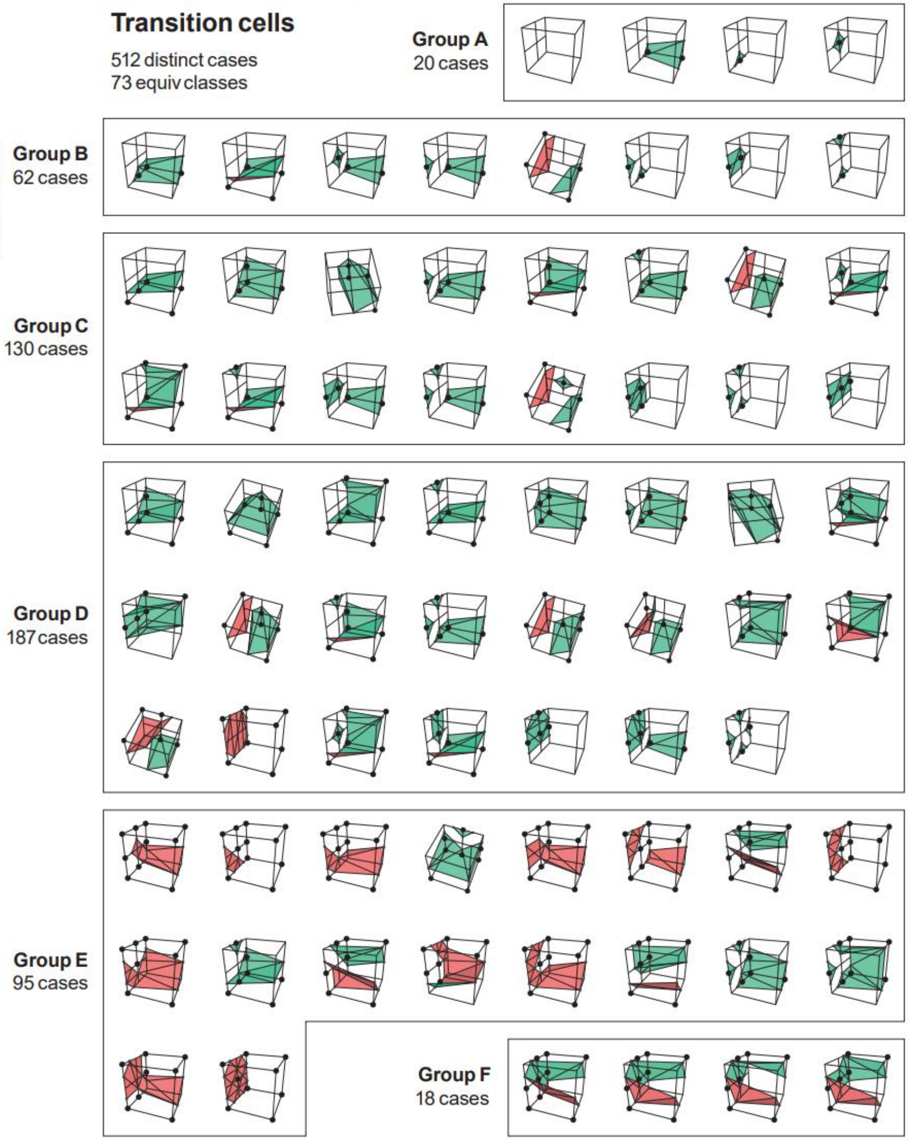
Transition Cell Lookup Table

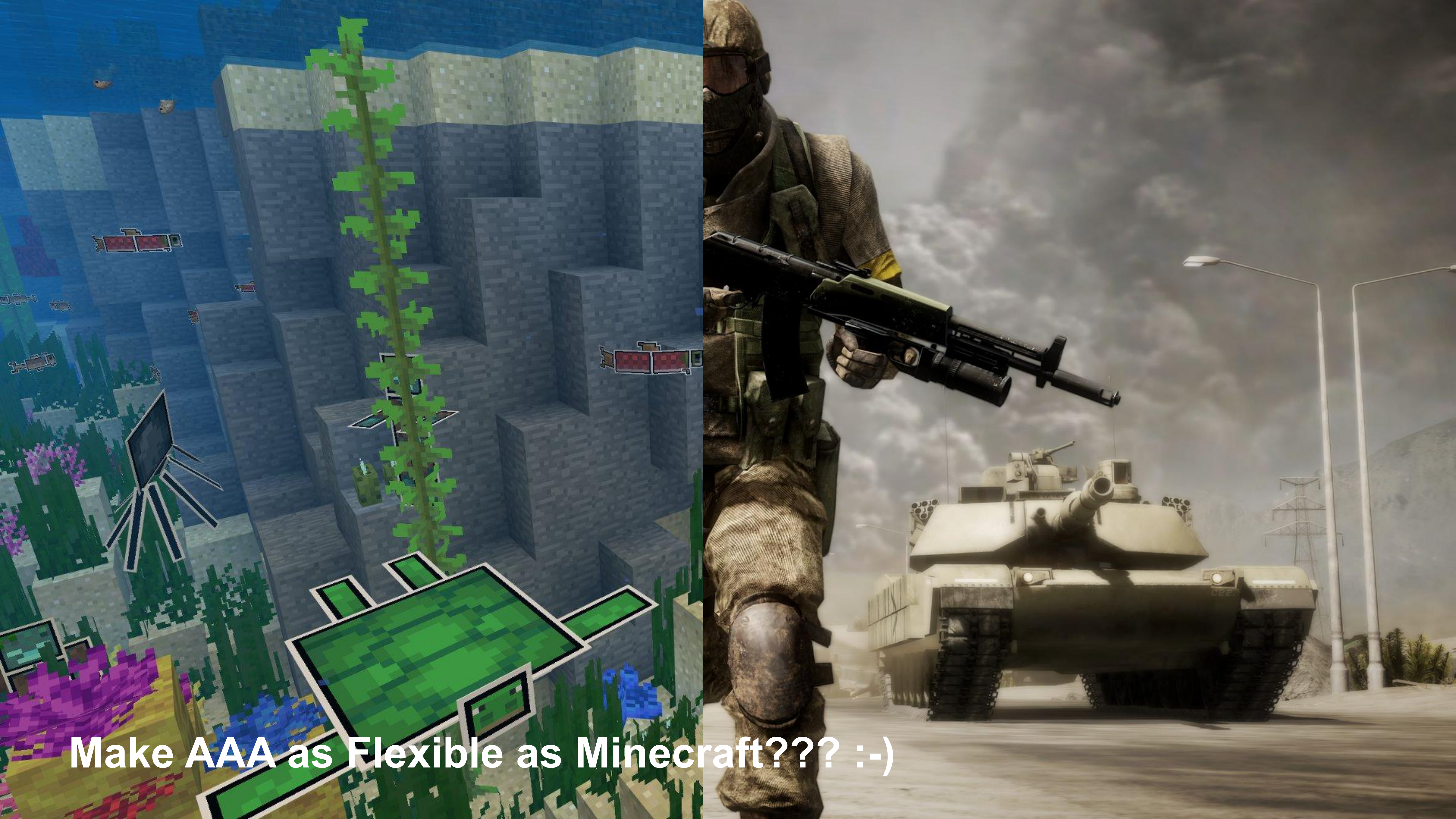
Transvoxel Algorithm

- Constructs the triangulation of transition cells to form a lookup table, and uses this lookup table to do the triangulation of LOD voxel cubes



Lengyel, Eric. (2010). Voxel-Based Terrain for Real-Time Virtual Simulations.

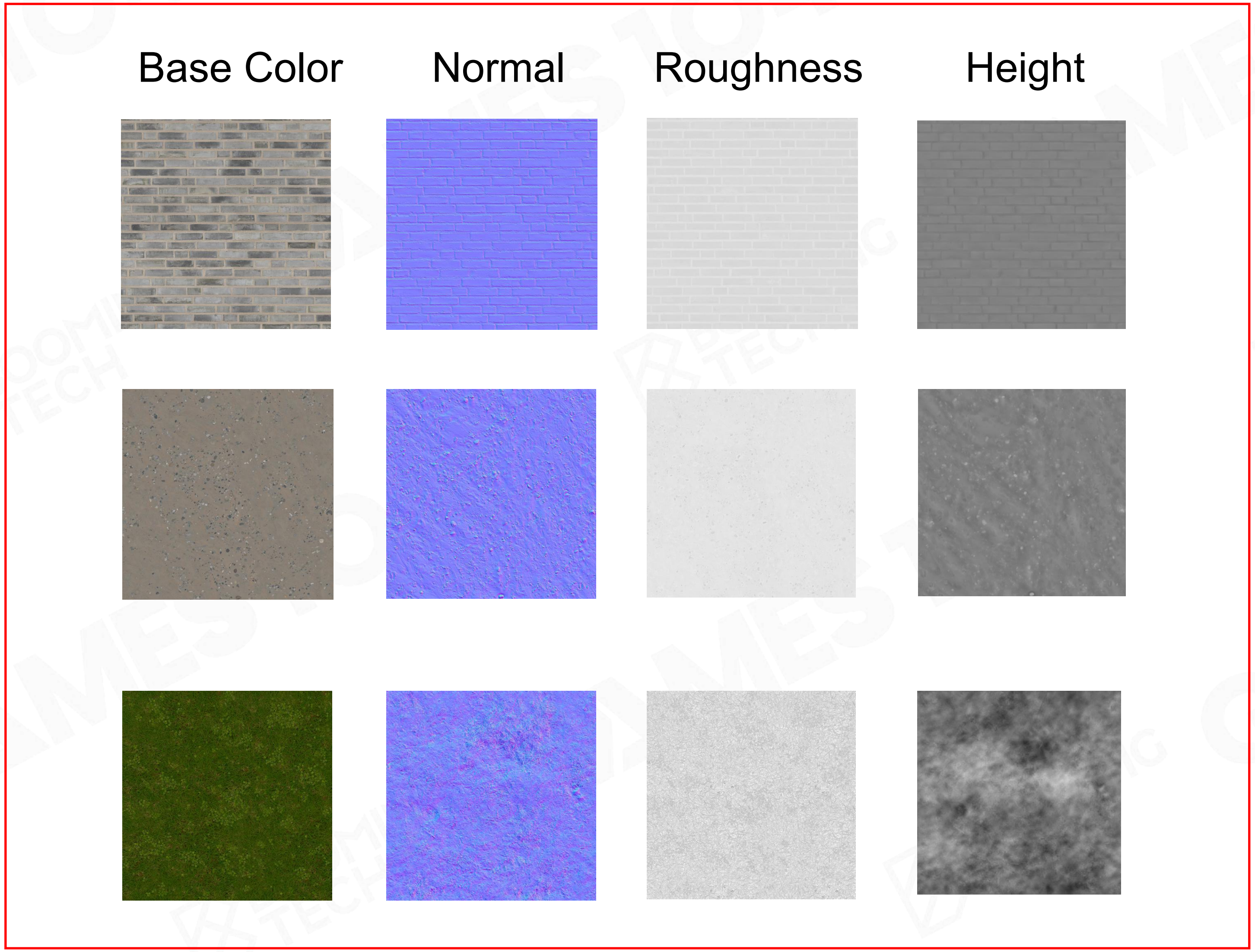


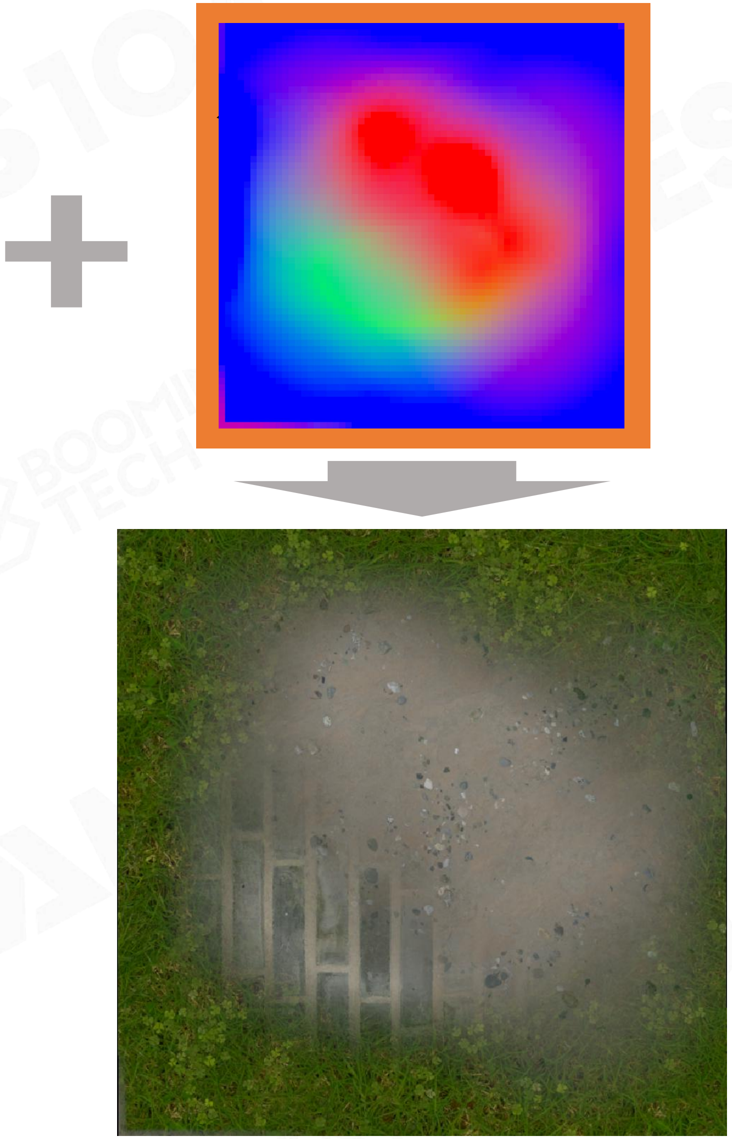

Make AAA as Flexible as Minecraft??? :-)


Paint Terrain Materials


11 Biomes 140 Materials
Ghost Recon Wildlands
11 Biomes
140 materials

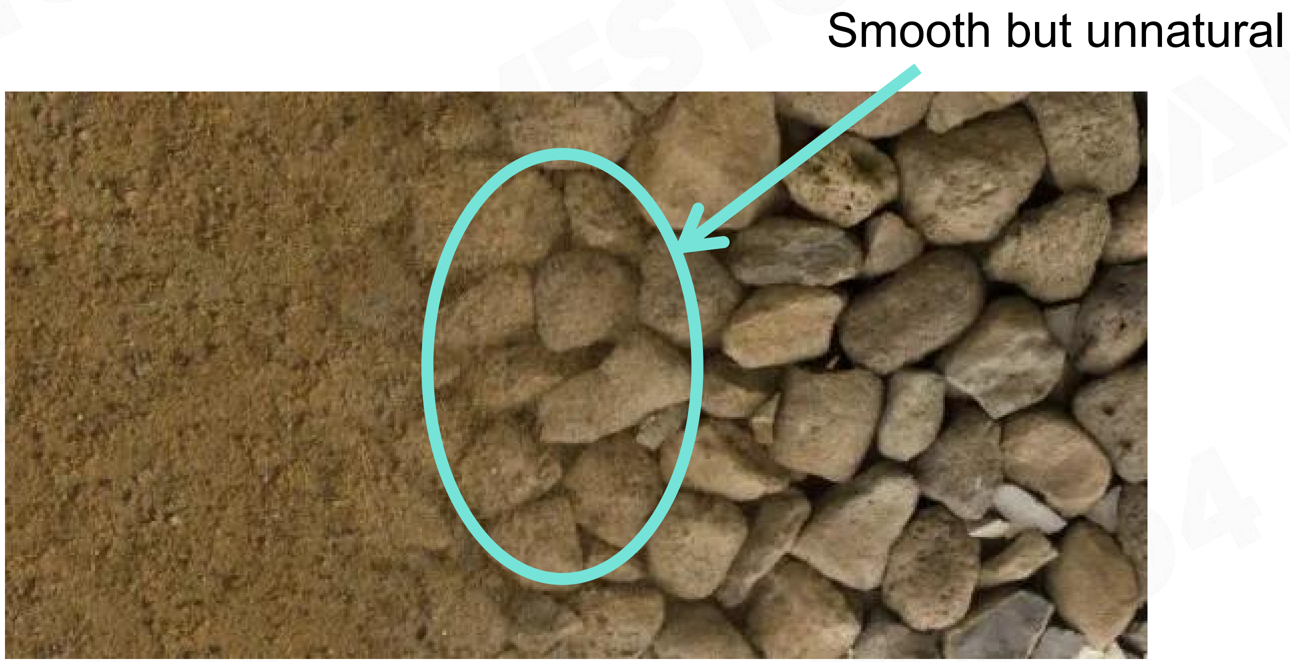
Terrain Materials





Material blending result

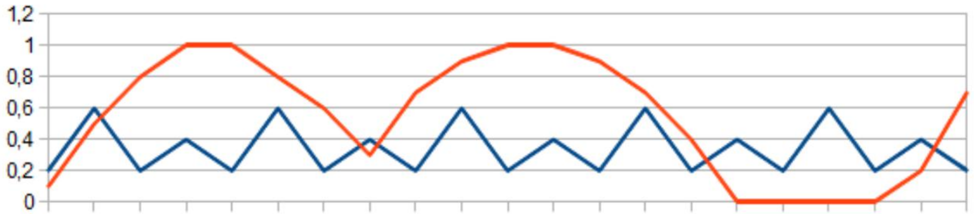
Simple Texture Splatting



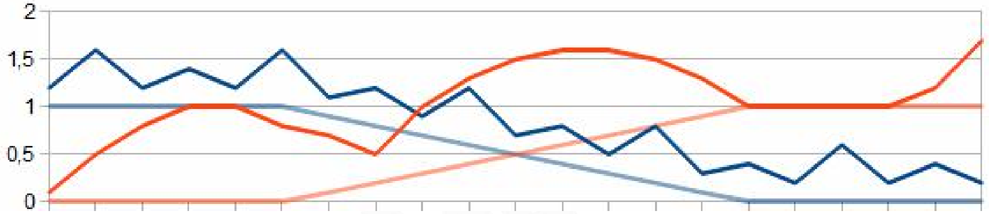
Simple Blending

```
float3 blend(float4 texture1, float a1, float4 texture2, float a2)
{
    return texture1.rgb * a1 + texture2.rgb * a2;
}
```

Advanced Texture Splatting



Height Maps

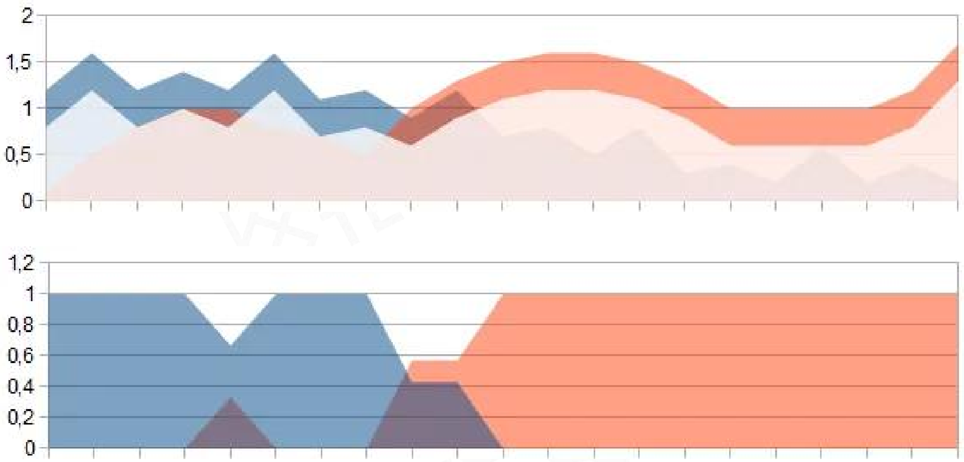


Height Maps + Alpha Blending

Blending with Height

```
float3 blend(float4 texture1, float height1, float4 texture2, float height2)
{
    return height1 > height2 ? texture1.rgb : texture2.rgb;
}
```

Advanced Texture Splatting - Biased

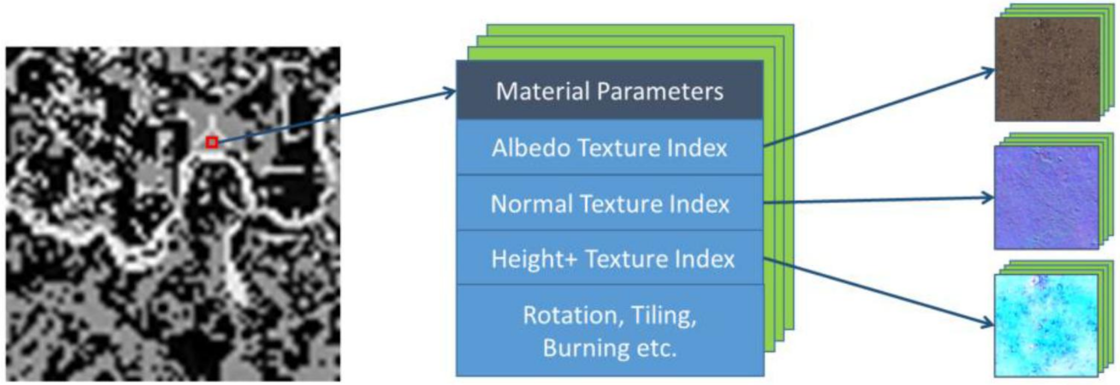


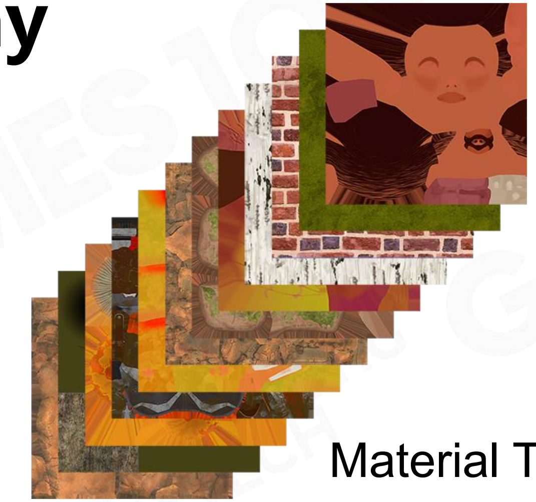

```
float3 blend(float4 texture1, float height1, float4 texture2, float height2)
{
    float depth = 0.2;
    float ma = max(texture1.a + height1, texture2.a + height2) - depth;
    float b1 = max(texture1.a + height1 - ma, 0);
    float b2 = max(texture2.a + height2 - ma, 0);
    return (texture1.rgb * b1 + texture2.rgb * b2) / (b1 + b2);
}
```

Height Bias

Links:
https://www.gamedeveloper.com/programming/advanced-terrain-texture-splatting

Sampling from Material Texture Array

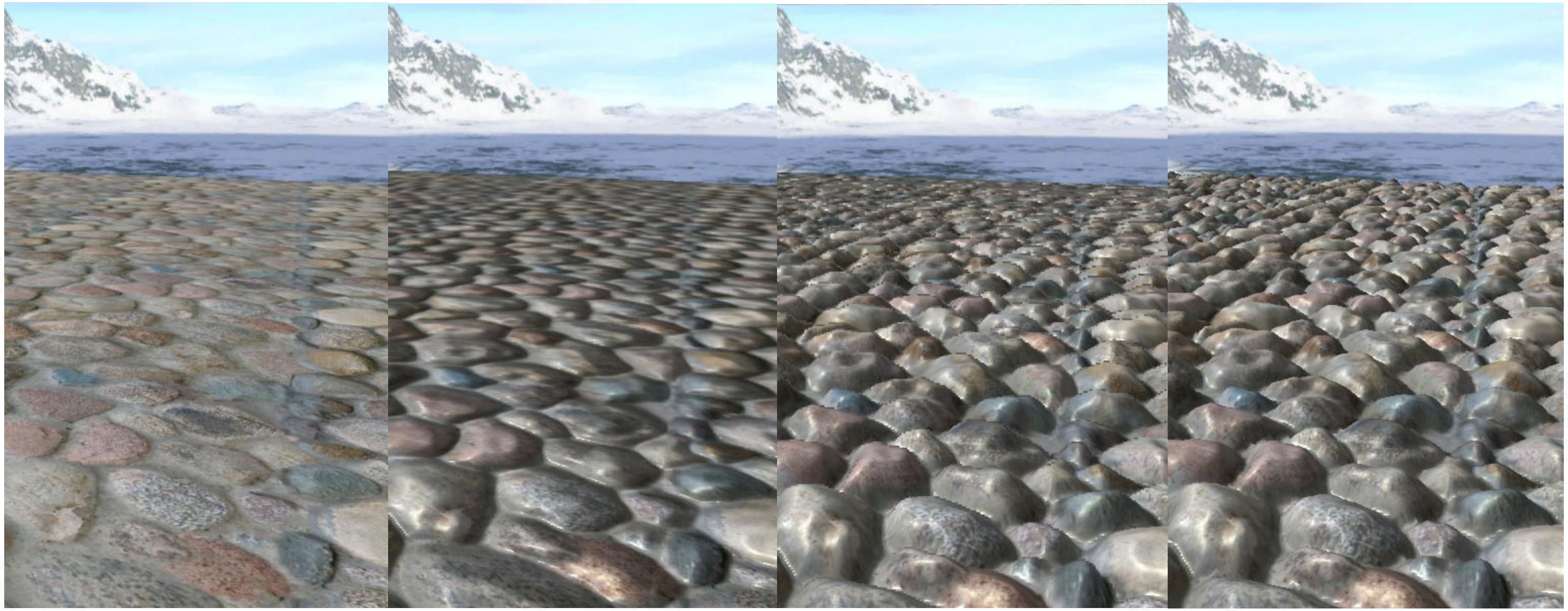




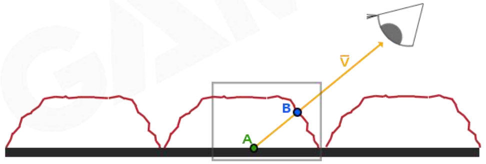
Material Texture Array

```
...

// get material index from splat map, and sample several directions of material for blending
int base_material_index    = sampleSplatMap(base_material_uv);
int right_material_index   = sampleSplatMap(right_material_uv);
int up_material_index      = sampleSplatMap(up_material_uv);
int rightup_material_index = sampleSplatMap(rightup_material_uv);

// get material parameters
float4 base_albedo = sampleAlbedoMapArray(base_map_uv, base_material_index);
float3 base_normal = sampleNormalMapArray(base_map_uv, base_material_index);
float  base_height = sampleHeightMapArray(base_map_uv, base_material_index);

float4 right_albedo = ...
float3 right_normal = ...
float  right_height = ...

...

// get blend weights according to material heights and use bilinear interpolation
float blend_weights = getBlendWeights(base_height, right_height, up_height, rightup_height);

// blend base, up, right, rightup layer data to get a smooth shading result
float4 blend_albedo =
    blendAlbedoWithWeights(blend_weights, base_albedo, right_albedo, up_albedo, rightup_albedo);
float3 blend_normal =
    blendNormalWithWeights(blend_weights, base_normal, right_normal, up_normal, rightup_normal);
...
```

Parallax and Displacement Mapping



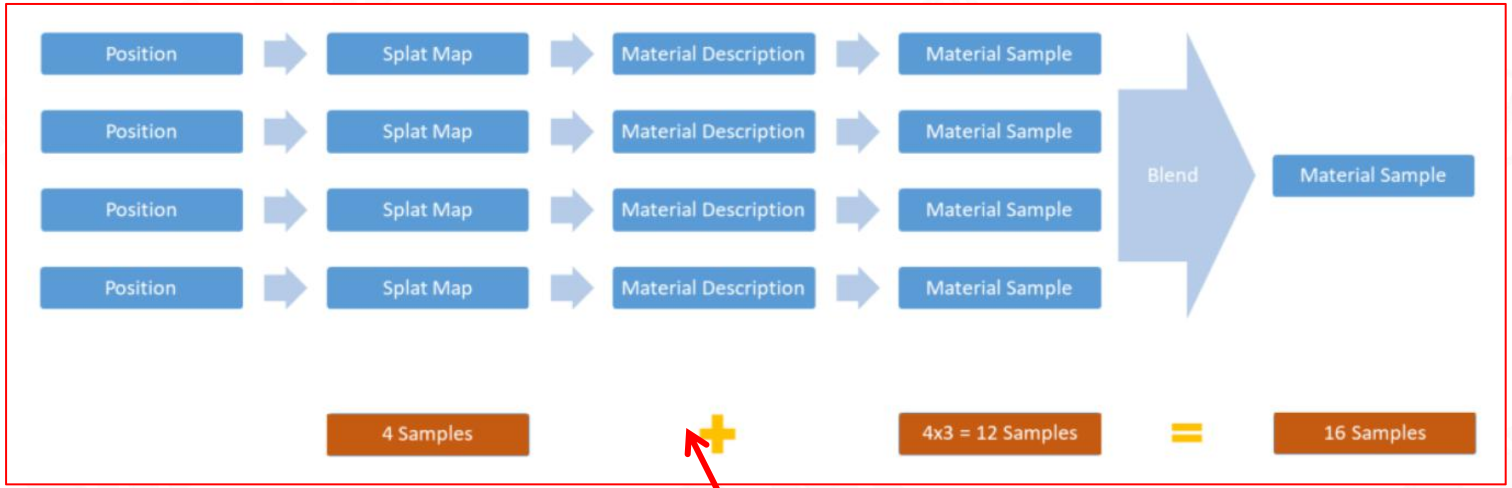
Color mapping | Bump mapping | Parallax mapping | Displacement mapping

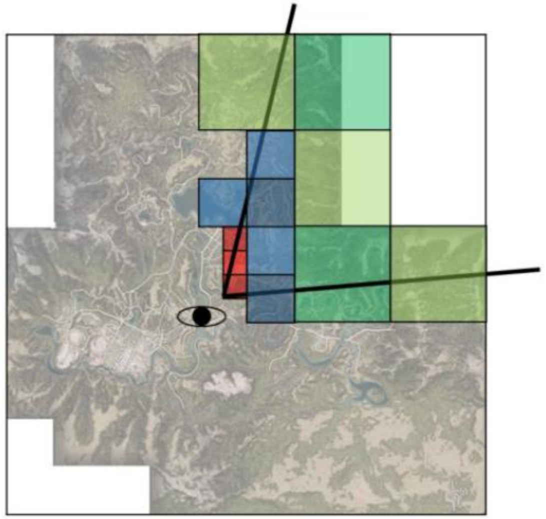


Parallax Mapping:
Due to the height of the surface, the eye sees point B instead of point A. It creates a sense of dimensionality

Expensive Material Blending

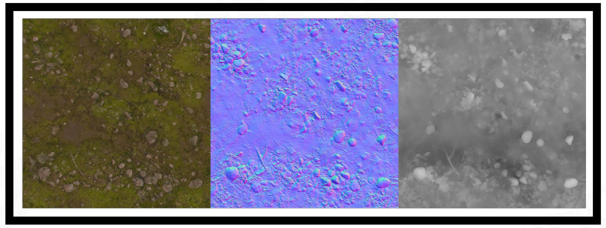
- **Many Texturing** - Low performance when multiple materials are sampled too many times
- **Huge Splat Map** - We only see a small set of terrain, but we load splat maps for 100 square km into video memory

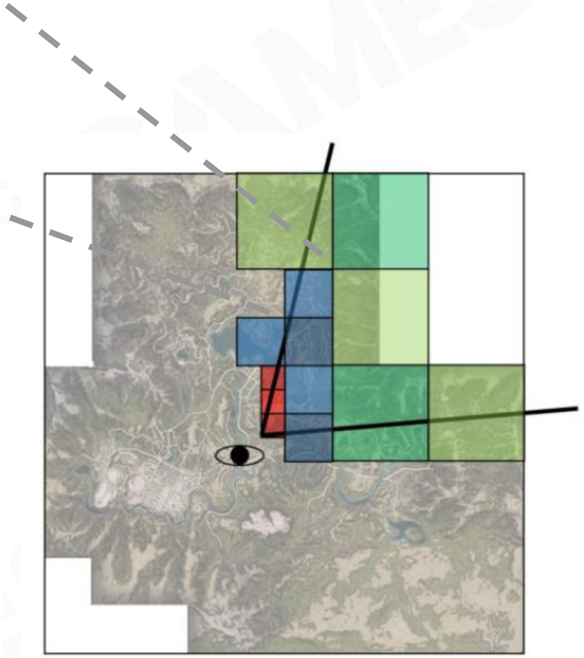


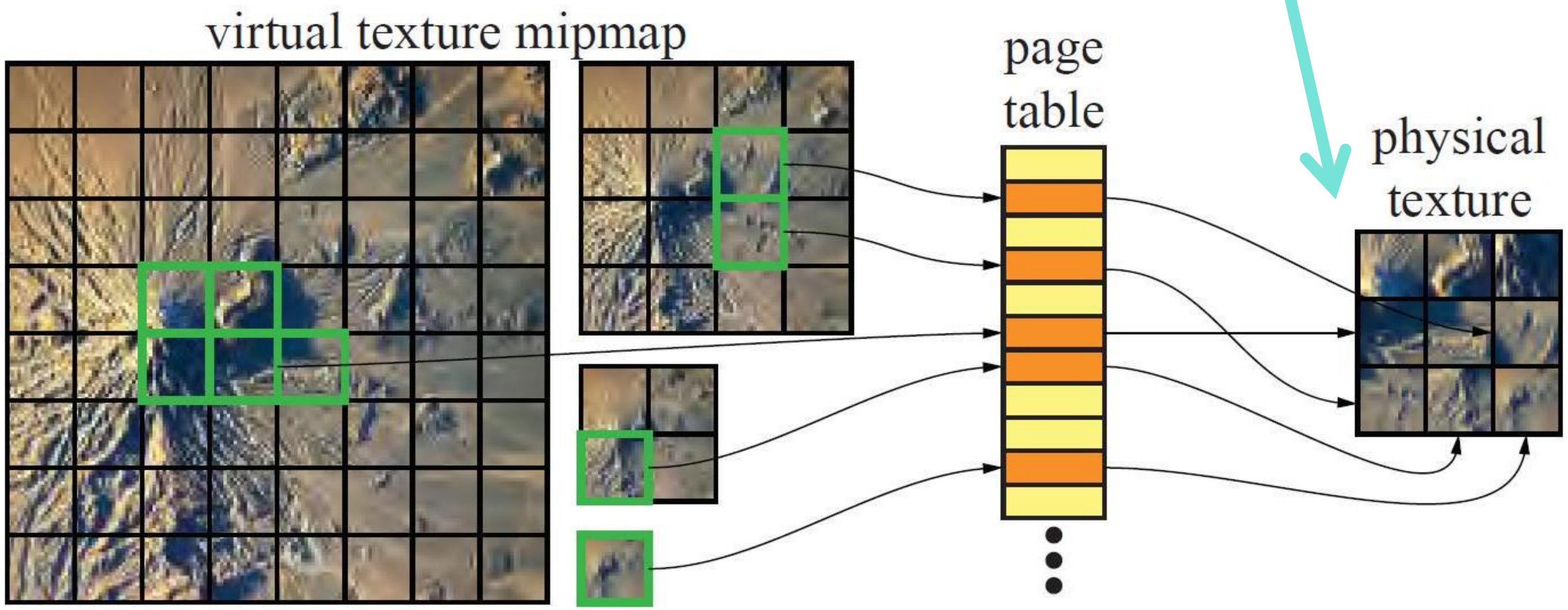


Virtual Texture

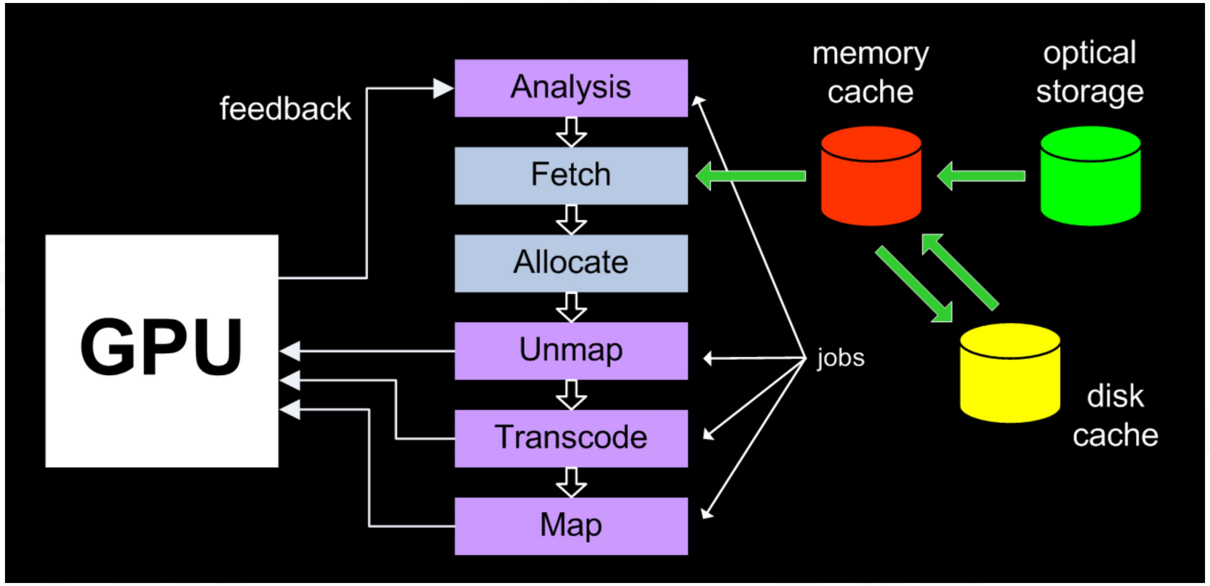
- Build a virtual indexed texture to represent all blended terrain materials for whole scene
- Only load materials data of tiles based on view-depend LOD
- Pre-bake materials blending into tile and store them into physical textures


Baked Terrain Tile

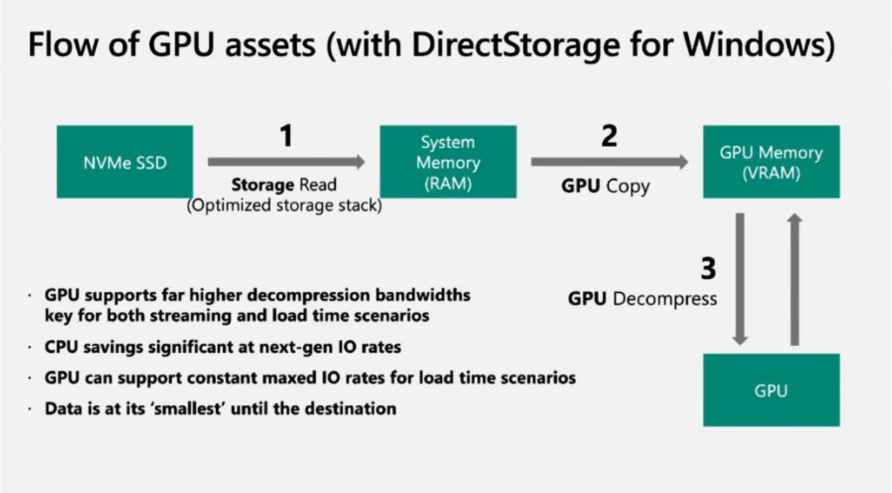




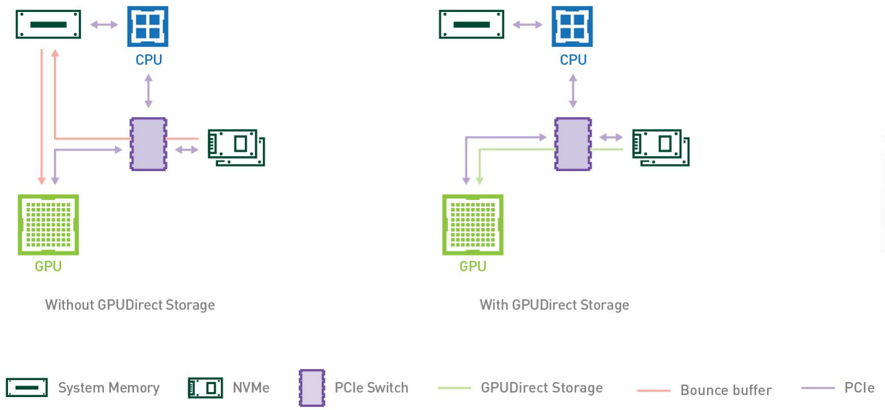
VT Implementation, DirectStorage & DMA



CPU based cache management among disk, main memory and video memory

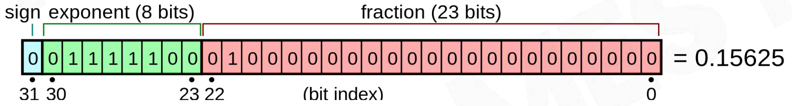


DirectStorage

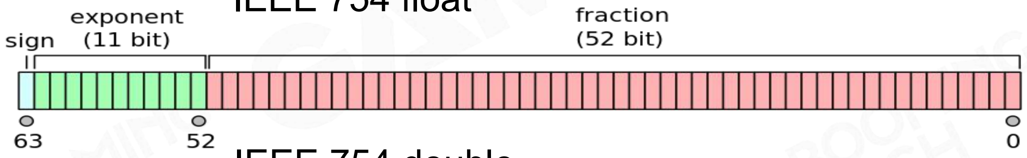


DMA

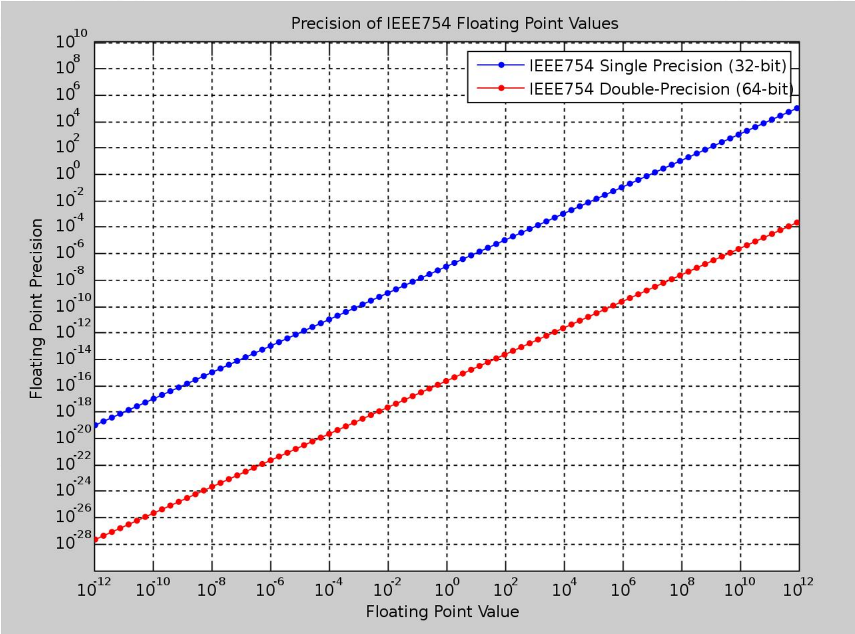
Floating-point Precision Error

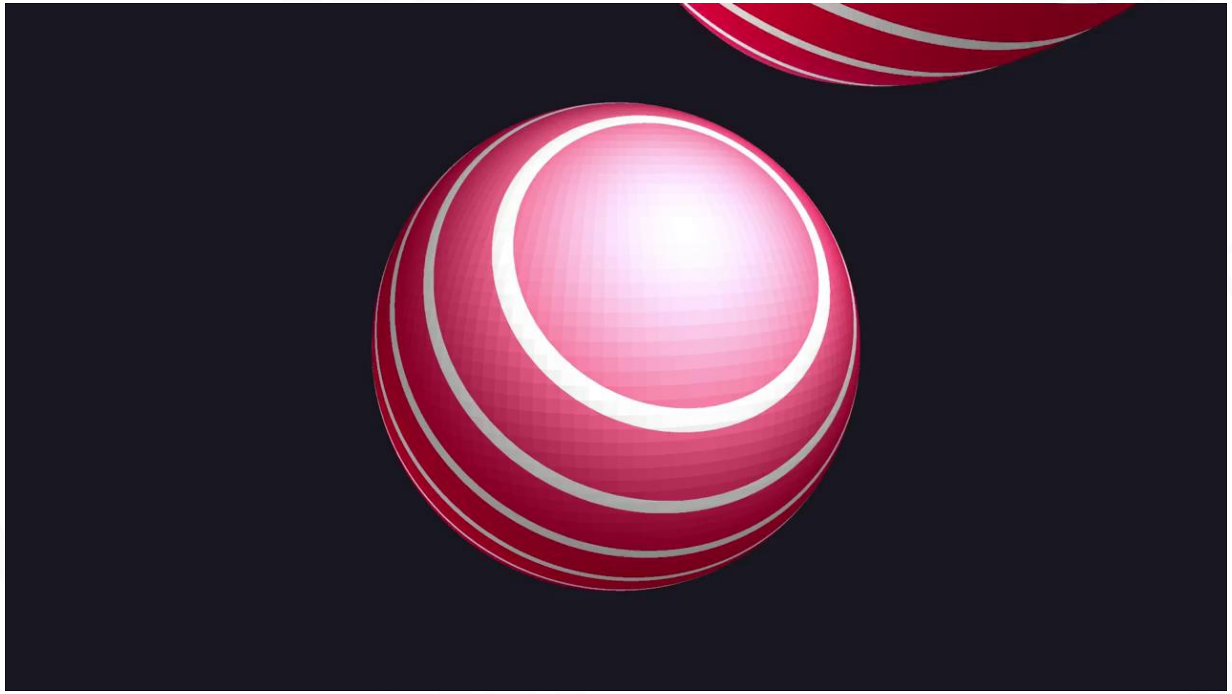


IEEE 754 float



IEEE 754 double



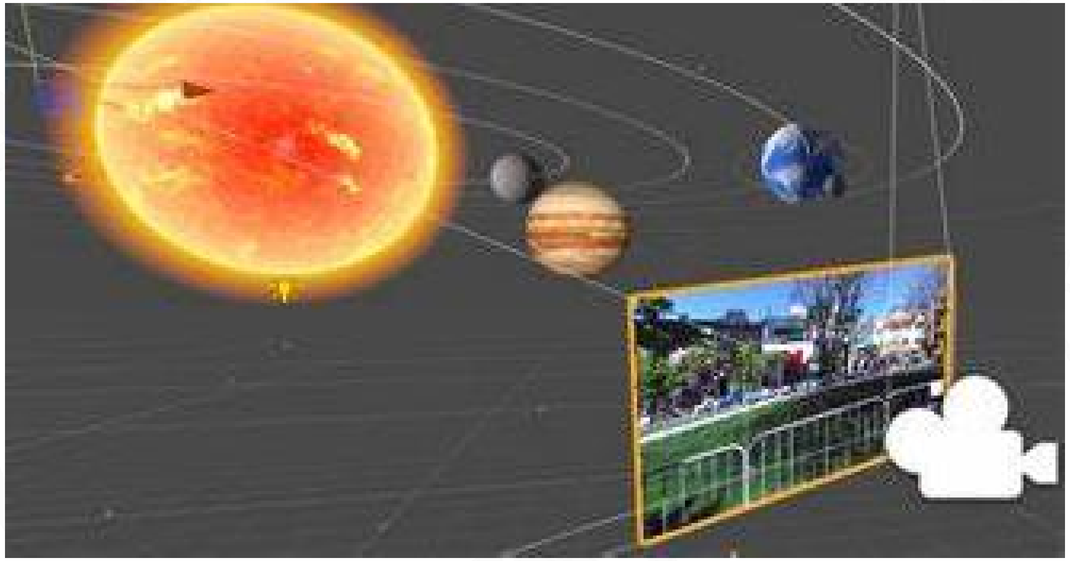


Floating-point error caused artifacts while camera and object in large value (from 1m to 60,000km)

Camera-Relative Rendering

- Translates objects by the negated world space camera position before any other geometric transformations affect them
- It then sets the world space camera position to 0 and modifies all relevant matrices accordingly

```
// camera relative
foreach render_object in render_objects
{
    render_object.m_position -= render_camera.m_position;
    updateRenderObjectTransform();
}

render_camera.m_position = Vector3(0.0, 0.0, 0.0);
updateRenderViewProjectionMatrix();
```



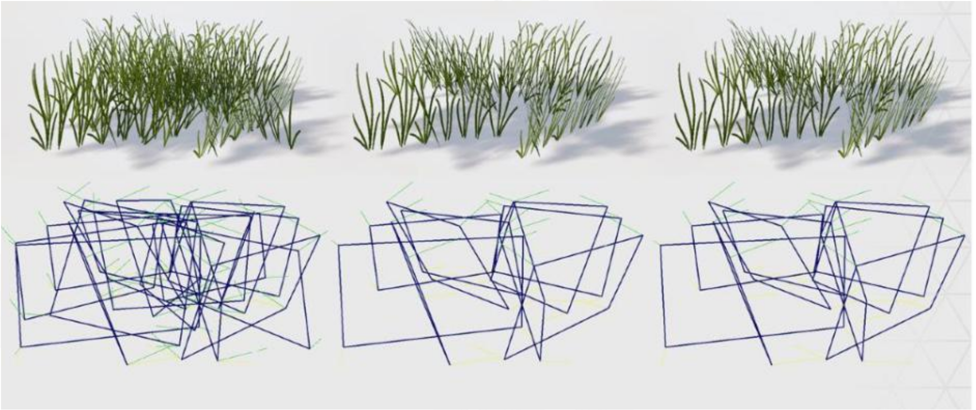
Render a whole galaxy :-)

Integration with other world elements (rocks, trees, grass)

Tree Rendering

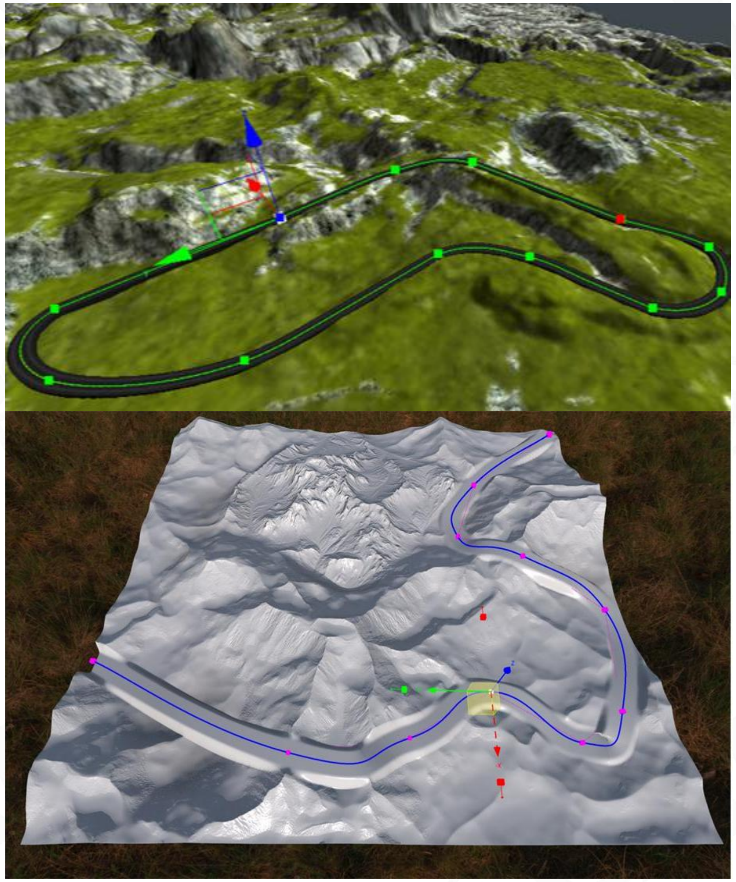
Tree Rendering LODs

Decorator Rendering



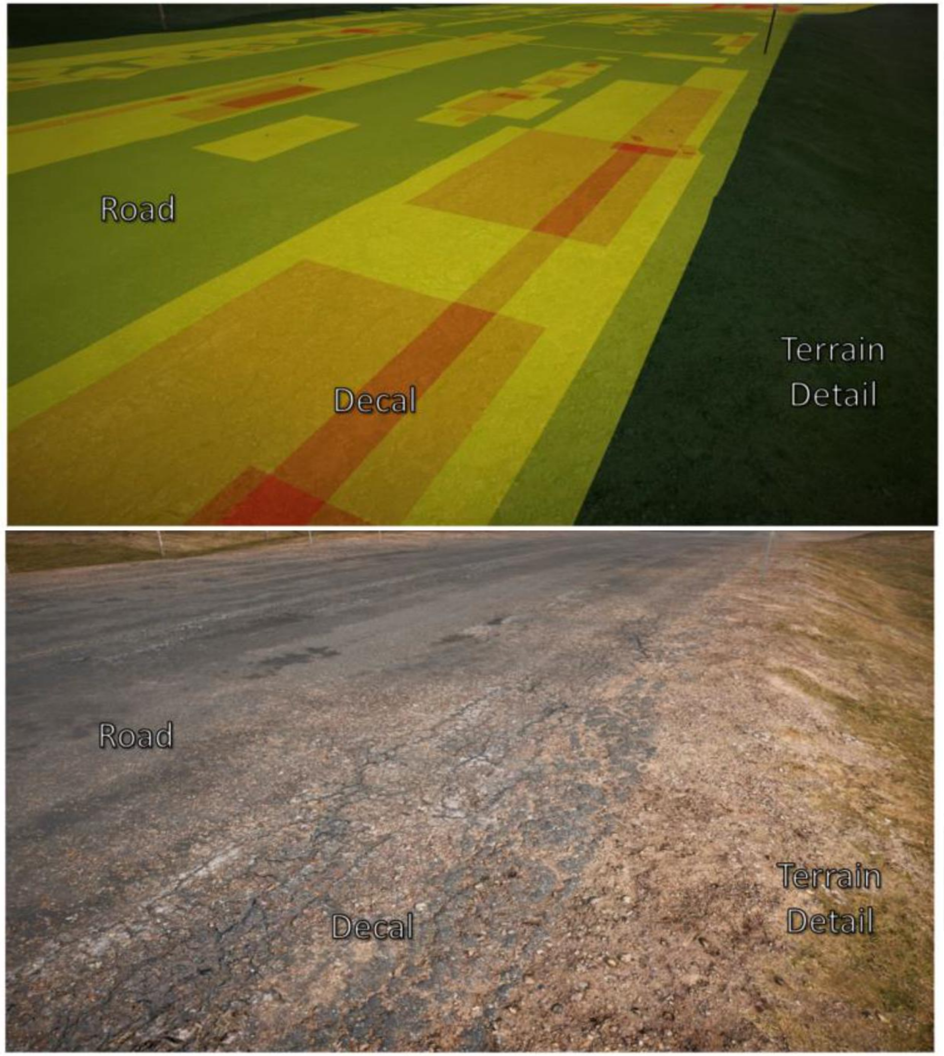
Decorator Rendering LODs

Road and Decals Rendering



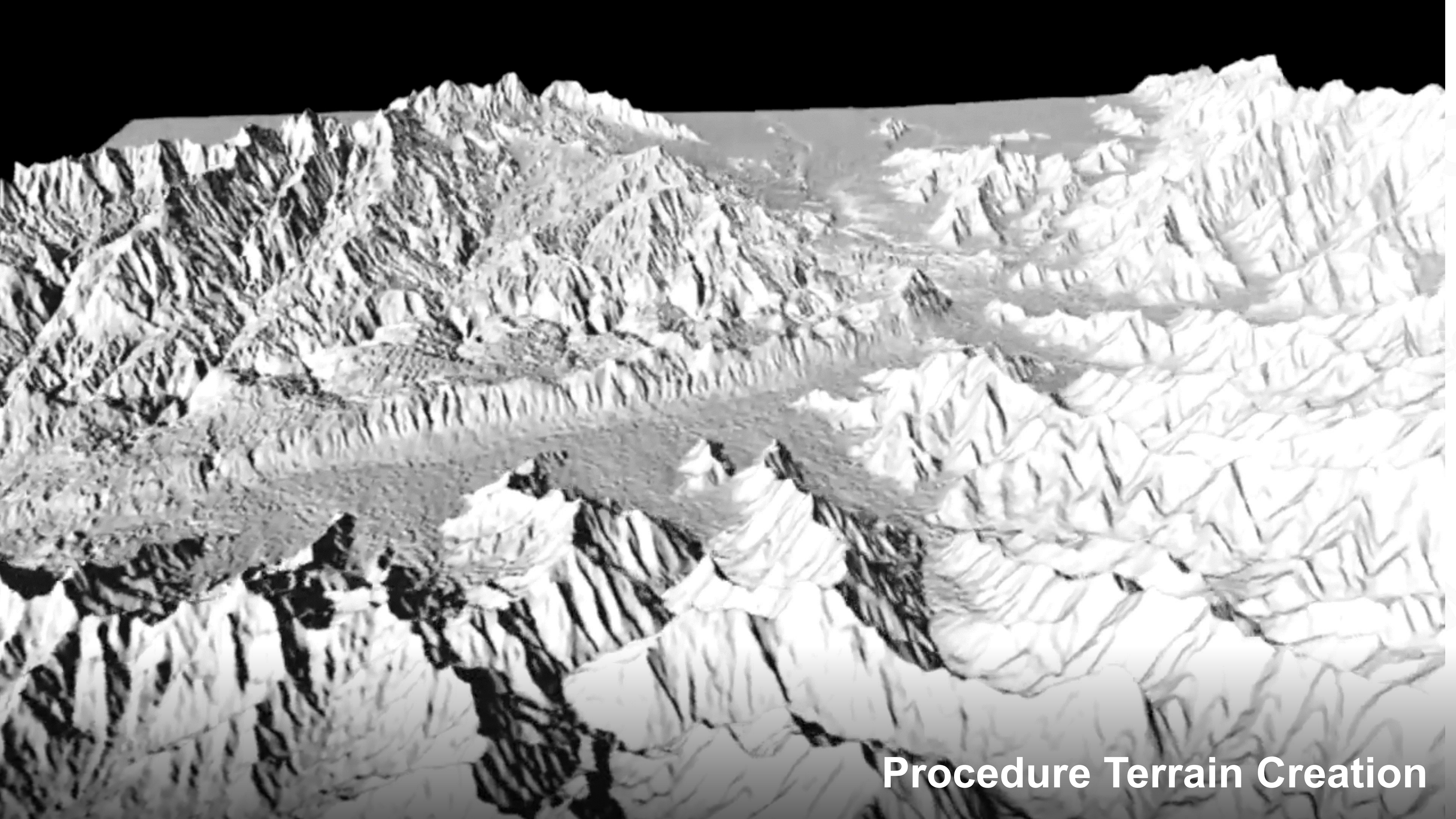
Spline-based Road Editing and Sculpturing Height Field

Decals



Splatting Road and Decals on Virtual Texture


File Editor View Tools Window Debug Setting Plugin Help
Default
Scale: 1
Place Actors
Tree Brush
Add plant
Scene
Level...
Debug
Terrai...
Terrai...
Import height map
Water level: 0
Altitude: 500
Select file: C:\Users\linjing.bao\Desktop\hei
Import
Import color map
BlockSelection
TileSelection
SetBaseBlock
Show Walkable Area
Show Selected Block
100
Adjust the selection area height: 0
Adjust
setWalkable
setUnWalkable
Operation
Mode: Draw Clear Snap
Target: All Selected
Brush
Radius: 28.43
Strength: 0.36
Tree Brush
MinScale: 80
MaxScale: 120
Content
Search path
Search
all
Terrain Editing in Game Engine


Procedure Terrain Creation

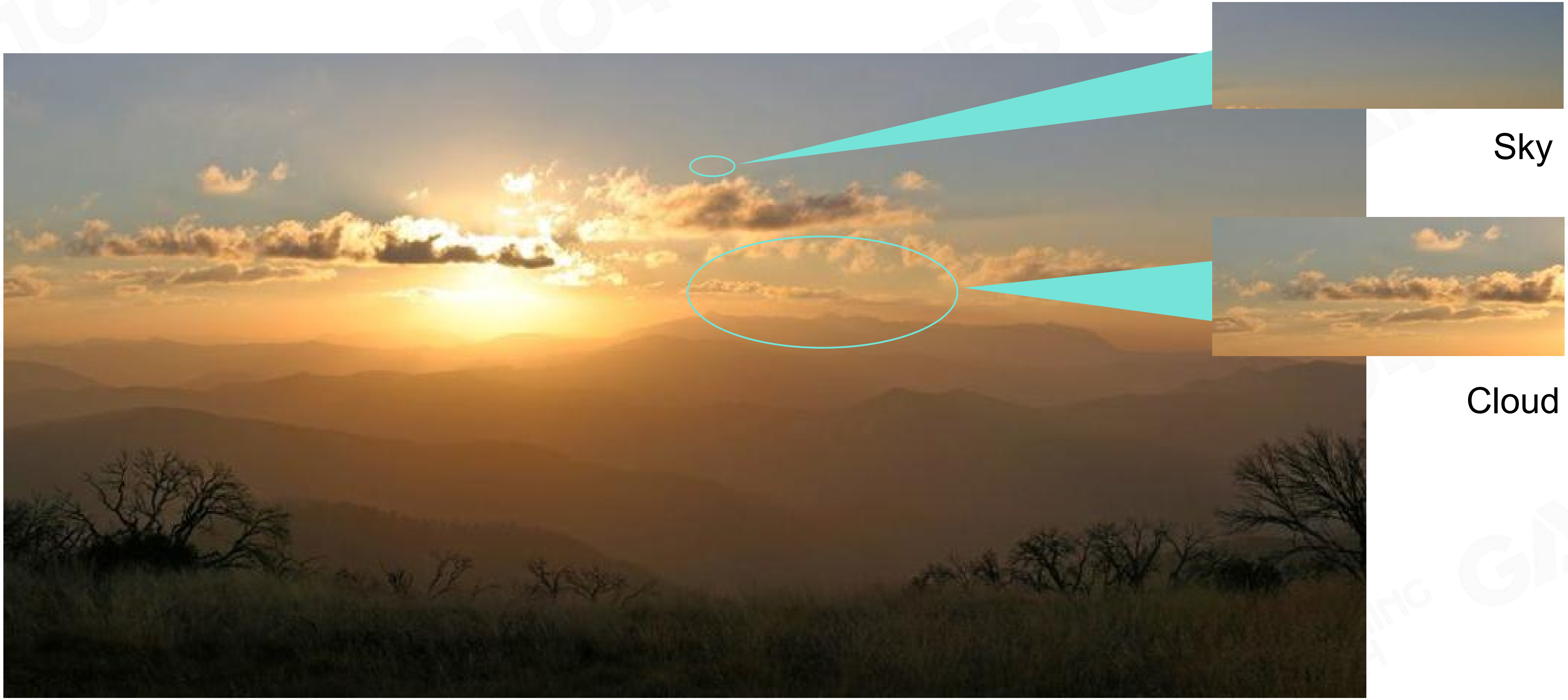
Sky and Atmosphere


Red Dead Redemption 2


HORIZON
II
FORBIDDEN WEST™
Captured on PS5™


ELDEN RING

How to "Paint" Everything in the Sky

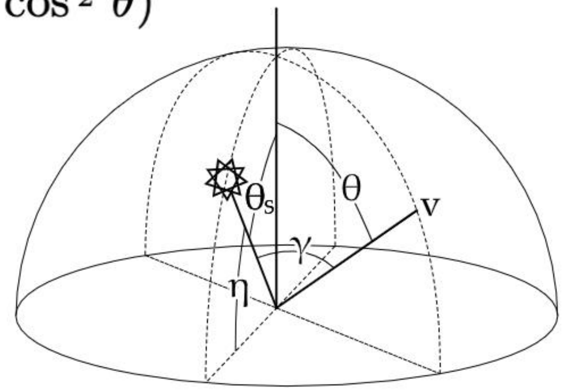


Atmosphere

Analytic Atmosphere Appearance Modeling

$$\mathbb{F}(\theta,\gamma) = (1 + Ae^{\frac{B}{\cos\theta + 0.01}}) \cdot (C + De^{E\gamma} + F\cos^2\gamma + G \cdot \chi(H,\gamma) + I \cdot \cos^{\frac{1}{2}}\theta)$$

$$L_\lambda = \mathbb{F}(\theta,\gamma) \cdot L_{M\lambda}$$



Pros

- Calculation is simple and efficient

Cons

- Limited to ground view
- Atmosphere parameters can't be changed freely

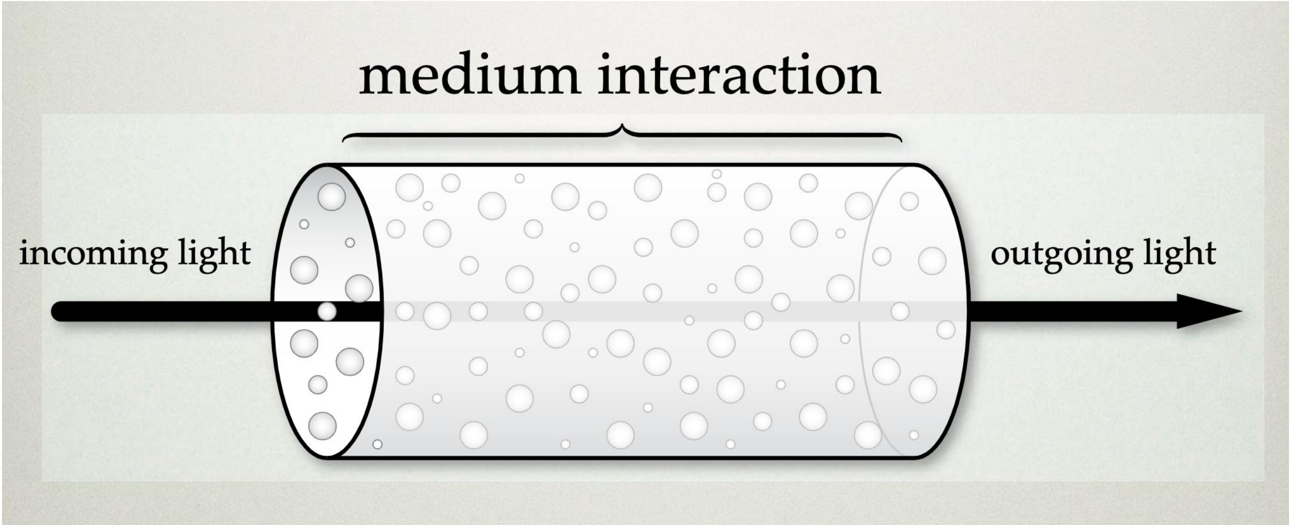
Photograph

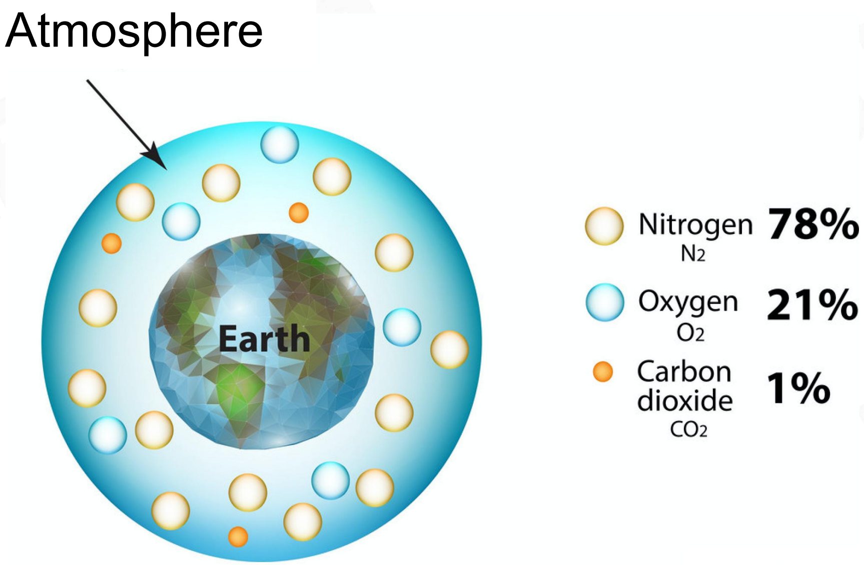
Rendering

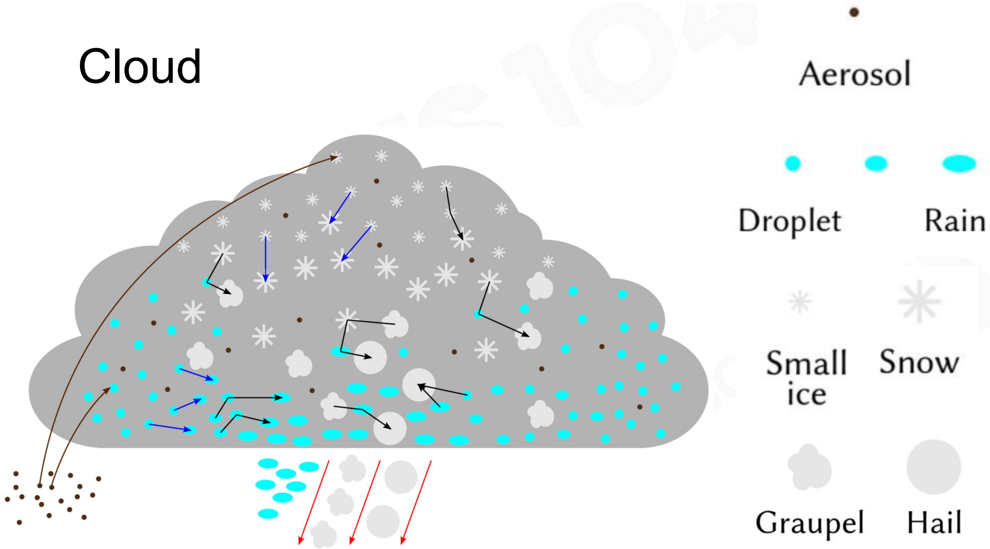
An Analytic Model for Full Spectral Sky-dome Radiance, ACM Trans 2012

Participating Media

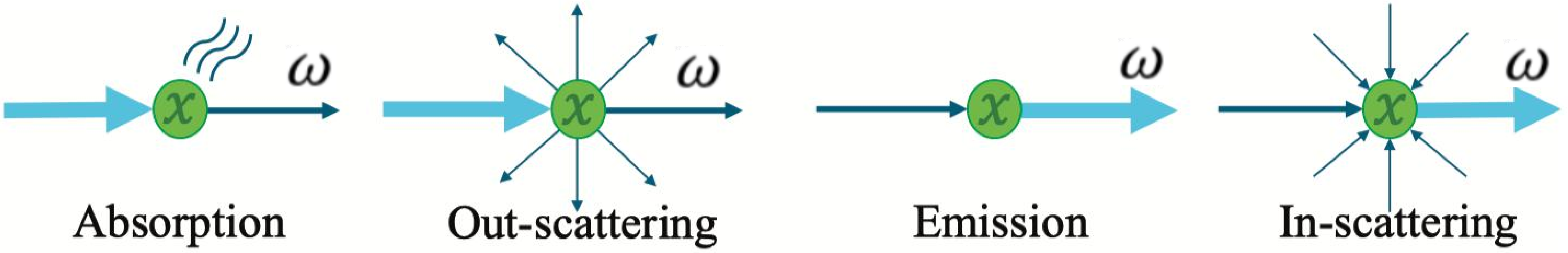
- Volume filled with particles
- Interact differently with light depending on its composition







How Light Interacts with Participating Media Particles?



$dL(x,\omega)/dx = -\sigma_a L(x,\omega)$ $\quad -\sigma_s L(x,\omega)$ $\quad \sigma_a L_e(x,\omega)$ $\quad \sigma_s \int_{S^2} f_p(x,\omega,\omega') L(x,\omega') d\omega'$

σ_a Absorption Coefficient $\quad \sigma_s$ Scattering Coefficient

Phase Function

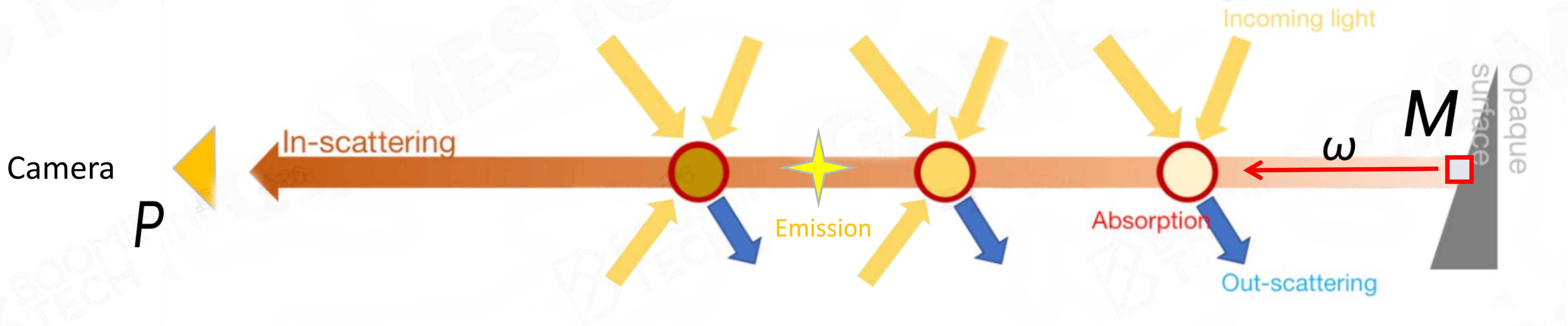
Radiative Transfer Equation (RTE)

Extinction Coefficient $\quad \sigma_t(x) = \sigma_a(x) + \sigma_s(x)$

$$dL(x,\omega)/dx = -\sigma_t L(x,\omega) + \sigma_a L_e(x,\omega) + \sigma_s \boxed{\int_{S^2} f_p(x,\omega,\omega') L(x,\omega') d\omega'}$$

In-Scattering Function

Volume Rendering Equation (VRE)



$$L(P,\omega) = \int_{x=0}^{d} T(x)[\sigma_a \cdot L_e(x,\omega) + \sigma_s \cdot L_i(x,\omega)]dx + T(M)L(M,\omega)$$

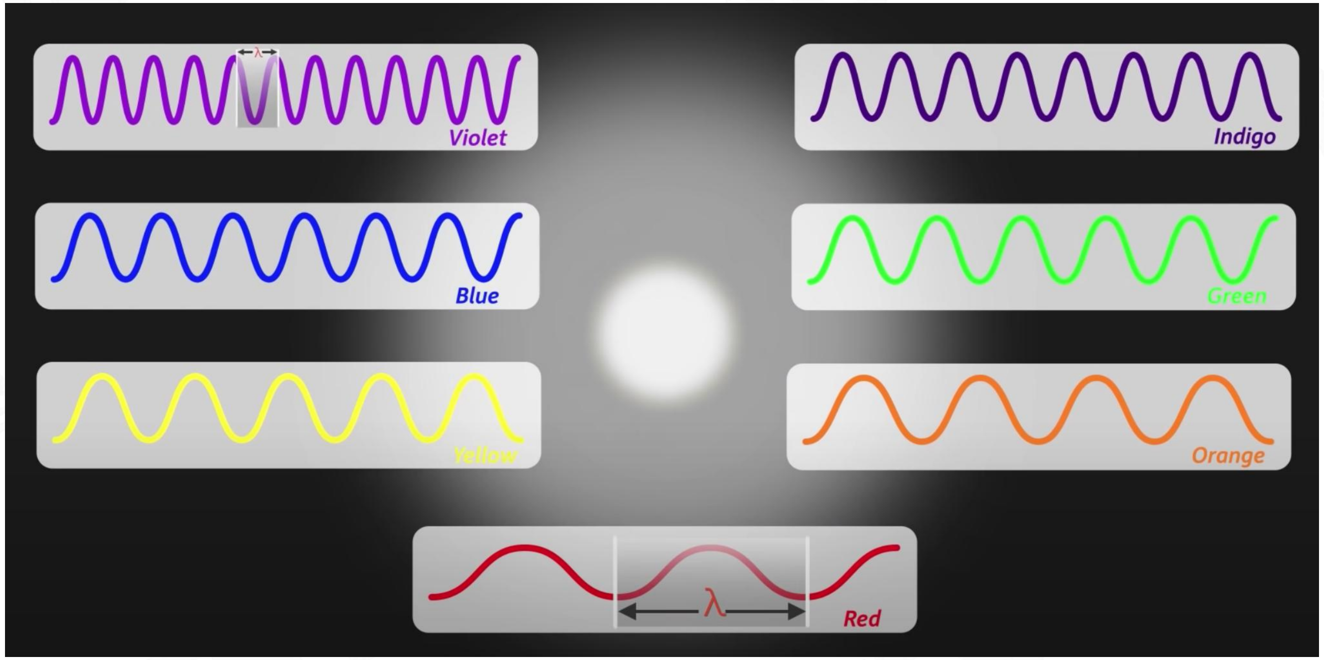
$$T(x) = e^{-\int_x^P \sigma_t(s)ds}$$

Transmittance: the net reduction factor from absorption and out-scattering

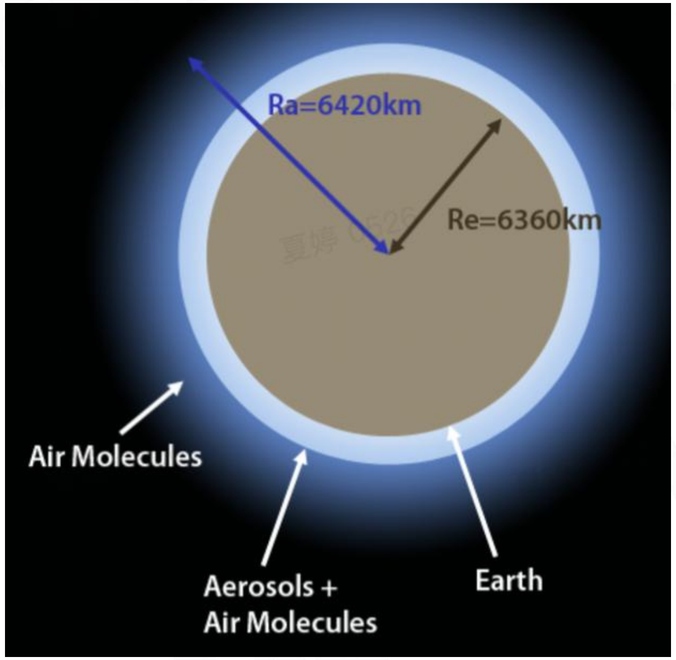
$$L_i(x,\omega) = \int_{S^2} f_p(x,\omega,\omega')L(x,\omega')d\omega'$$

The net increase factor from in-scattering

Real Physics in Atmosphere



Sun Light



- Air Molecules
  N2 O2 O3
- Aerosols
  Dust Sand

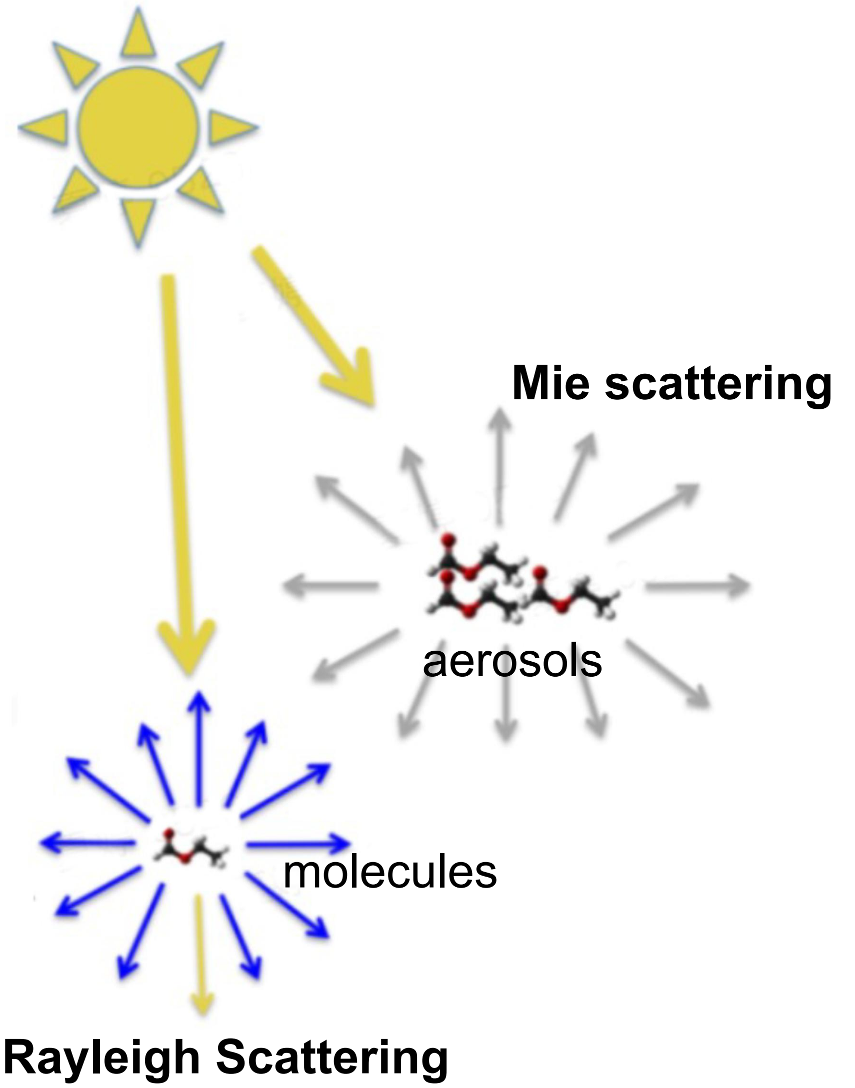
Scattering Types

- **Rayleigh Scattering**

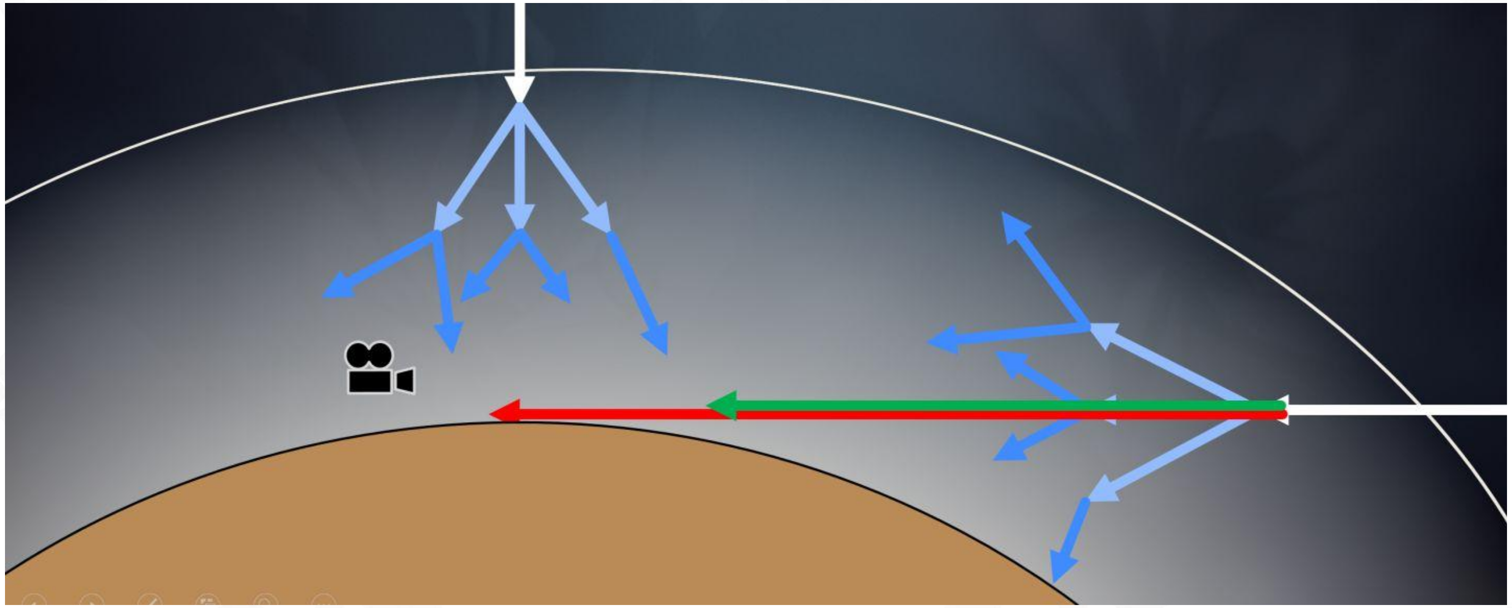
  Scattering of light by particles that have a diameter **much smaller than** the wavelength of the radiation (eg. air molecules）

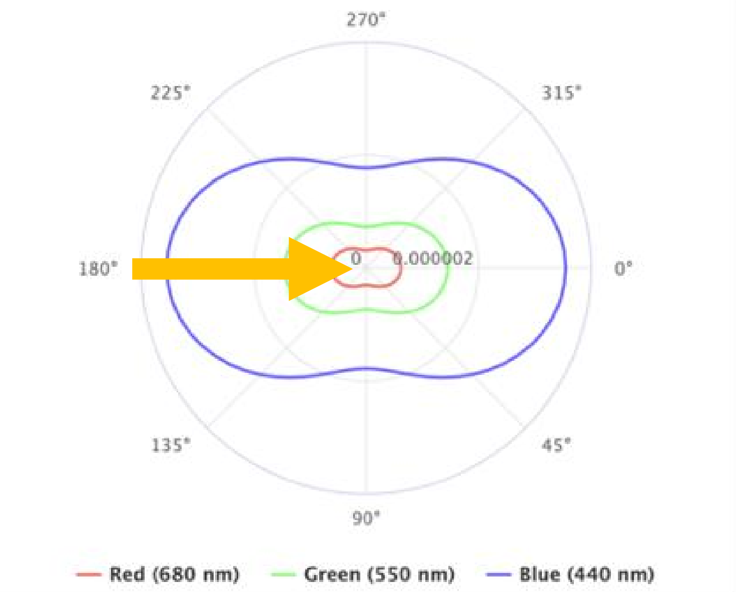
- **Mie scattering**

  Scattering of light by particles that have a diameter **similar to or larger than** the wavelength of the incident light (eg. aerosols)



Rayleigh Scattering





Rayleigh Scattering Distribution

- Certain directions receive more light than others
  front-back symmetry
- Shorter wavelengths (eg. blue) are scattered more
  strongly than longer wavelengths (eg. red)

Rayleigh Scattering Equation

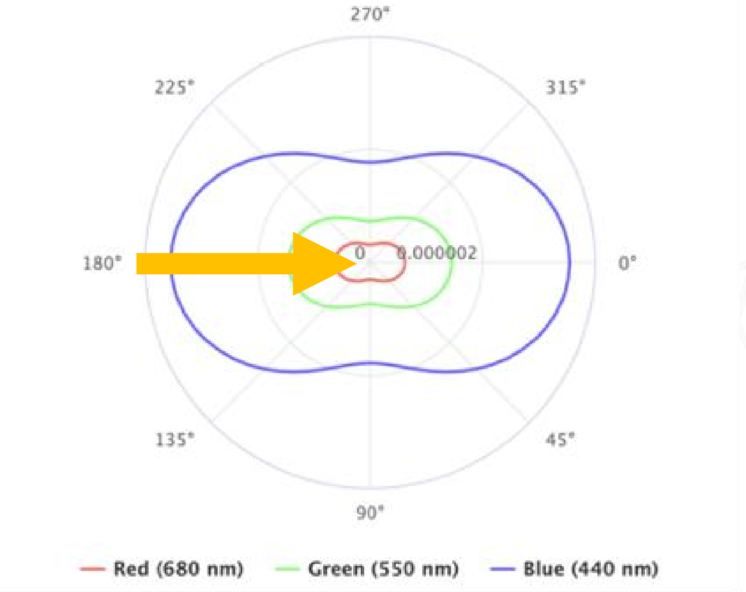
$$S(\lambda,\theta,h) = \frac{\pi^2(n^2-1)^2}{2}\underbrace{\frac{\rho(h)}{N}}_{\text{Density}}\overbrace{\frac{1}{\lambda^4}}^{\text{Wavelength}}\underbrace{(1+\cos^2\theta)}_{\text{Geometry}}$$

$$\sigma_s^{Rayleigh}(\lambda,h) = \frac{8\pi^3(n^2-1)^2}{3}\frac{\rho(h)}{N}\frac{1}{\lambda^4}$$

Scattering Coefficient

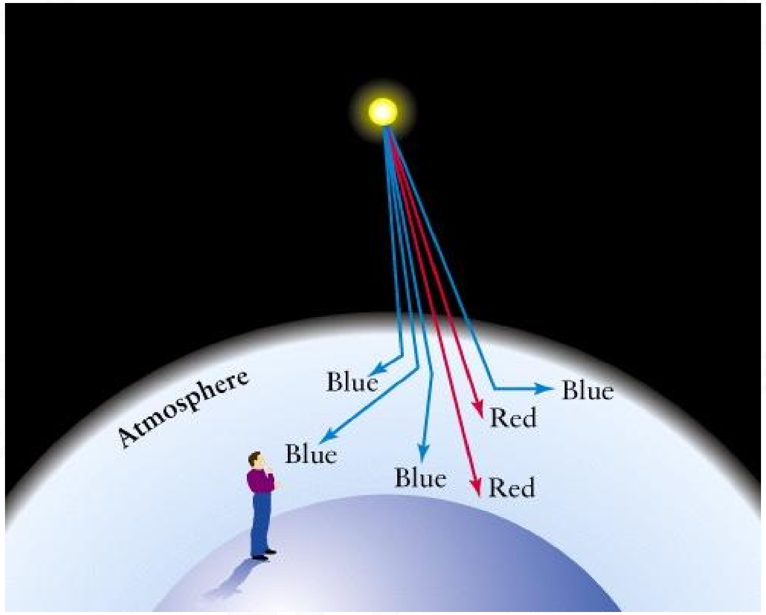
$$F_{Rayleigh}(\theta) = \frac{3}{16\pi}(1+\cos^2\theta)$$

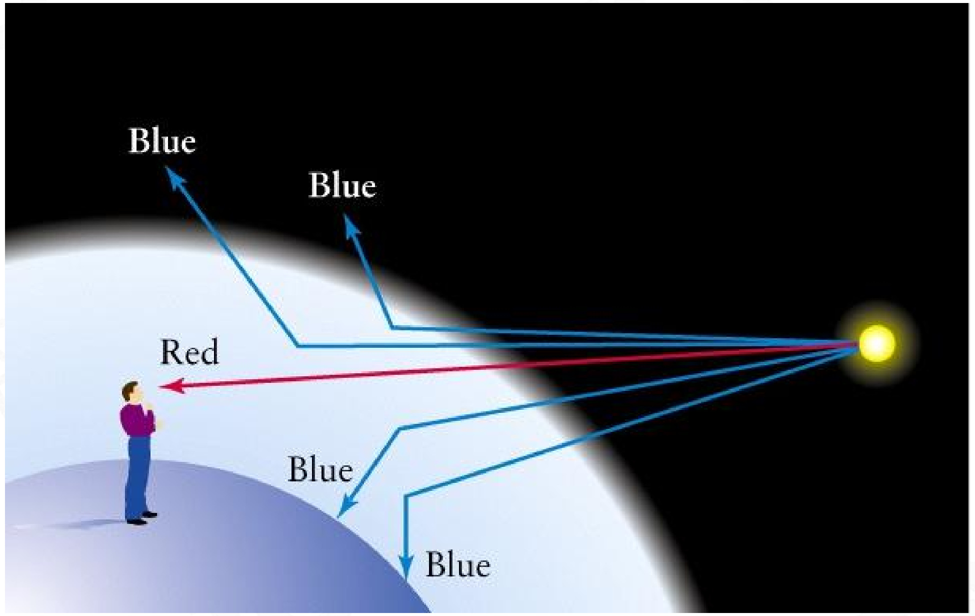
Phase Function



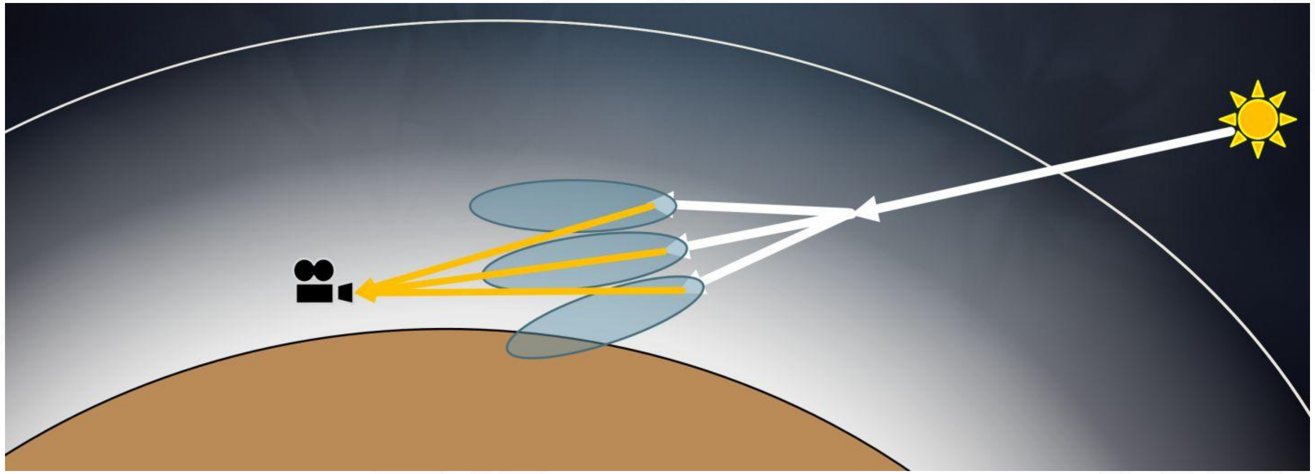
Rayleigh Scattering Distribution

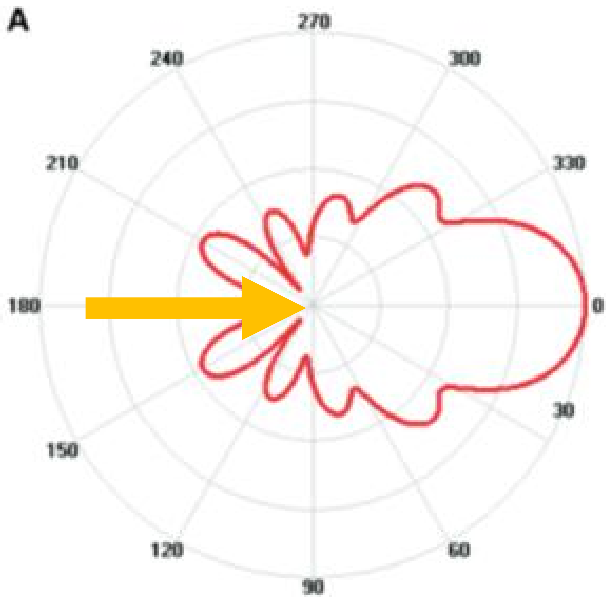
Why Sky is Blue





Mie Scattering





Mie Scattering Distribution

- Scatter light of all wavelength nearly equally
- Exhibit a strong forward directivity

Mie Scattering Equation

Geometry Parameter

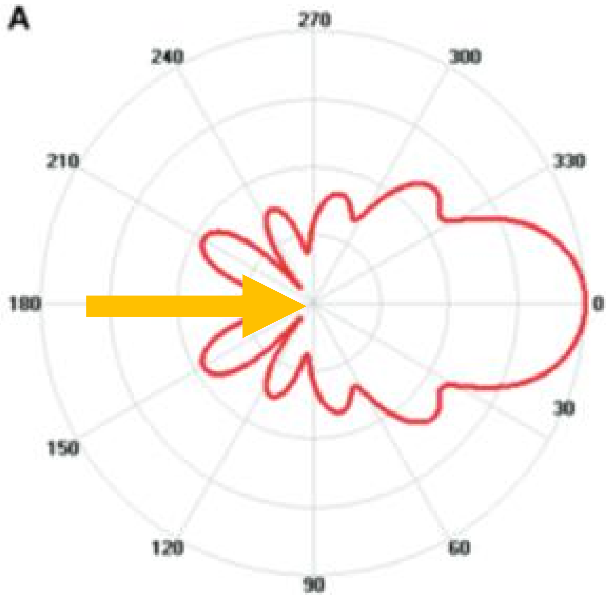
$$S(\lambda,\theta,h) = \pi^2(n^2-1)^2\frac{\rho(h)}{N}\frac{1-\boxed{g}^2}{2+g^2}\frac{1+cos^2\theta}{(1-g^2-2gcos\theta)^{\frac{3}{2}}}$$

$$\sigma_s^{Mie}(\lambda,h) = \frac{8\pi^3(n^2-1)^2}{3}\frac{\rho(h)}{N}$$

Scattering Coefficient

$$F_{Mie}(\theta) = \frac{3}{8\pi}\frac{1-g^2}{2+g^2}\frac{1+cos^2\theta}{(1-g^2-2gcos\theta)^{\frac{3}{2}}}$$

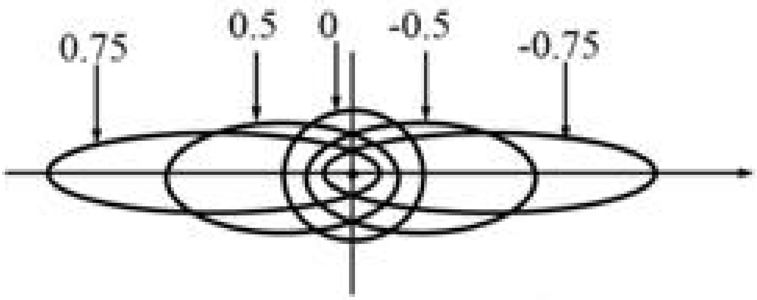
Phase Function



Mie Scattering Distribution

- g > 0, scatters more forward

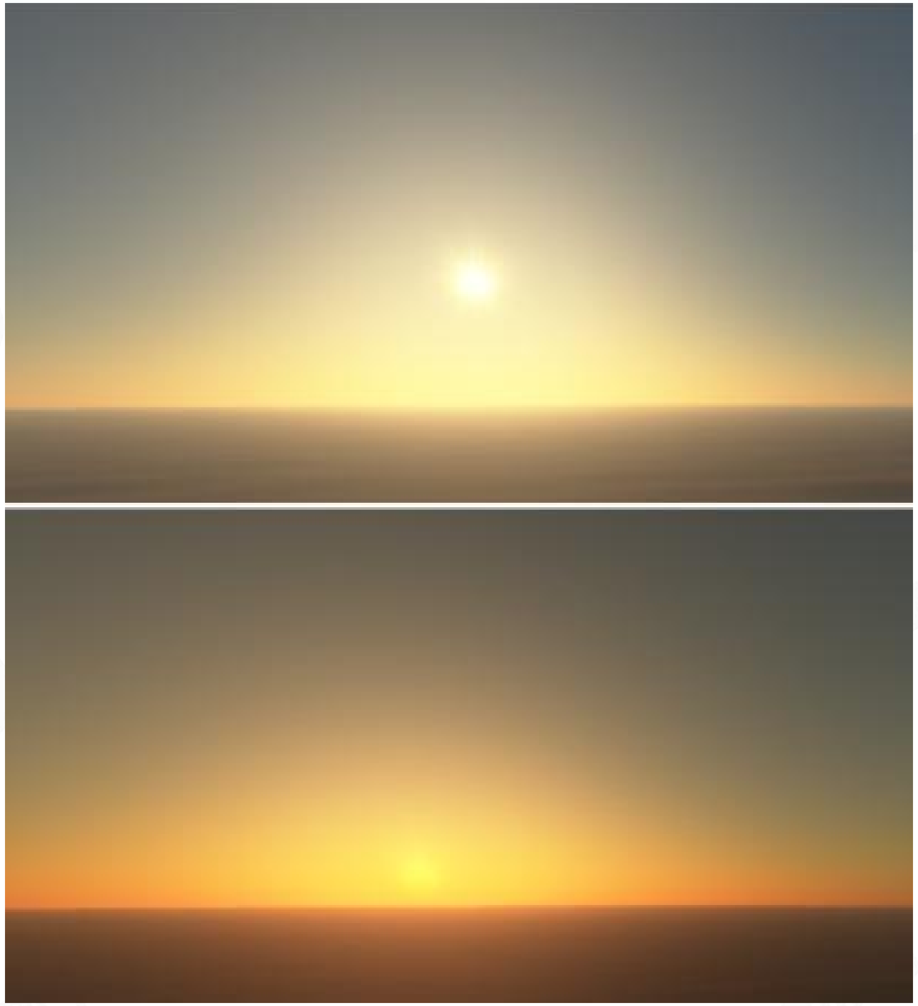
  Mie scattering
- g < 0, scatters more backward
- g = 0, Rayleigh scattering



Mie Scattering in Daily Life

- Exhibit a strong forward directivity (halo effects around sun)
- Scatter light of all wavelength nearly equally (fog effects）

Fog



Halo of Sun

Variant Air Molecules Absorption

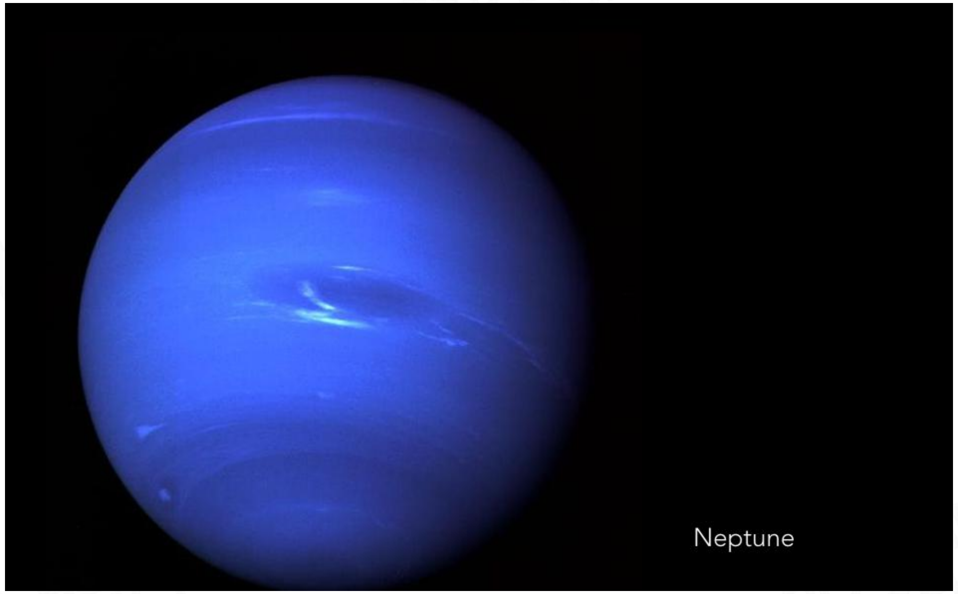
- Ozone (O_3)

  Absorb strongly at longer wavelengths to filter out the reds, oranges, yellows

- Methane (CH_4)

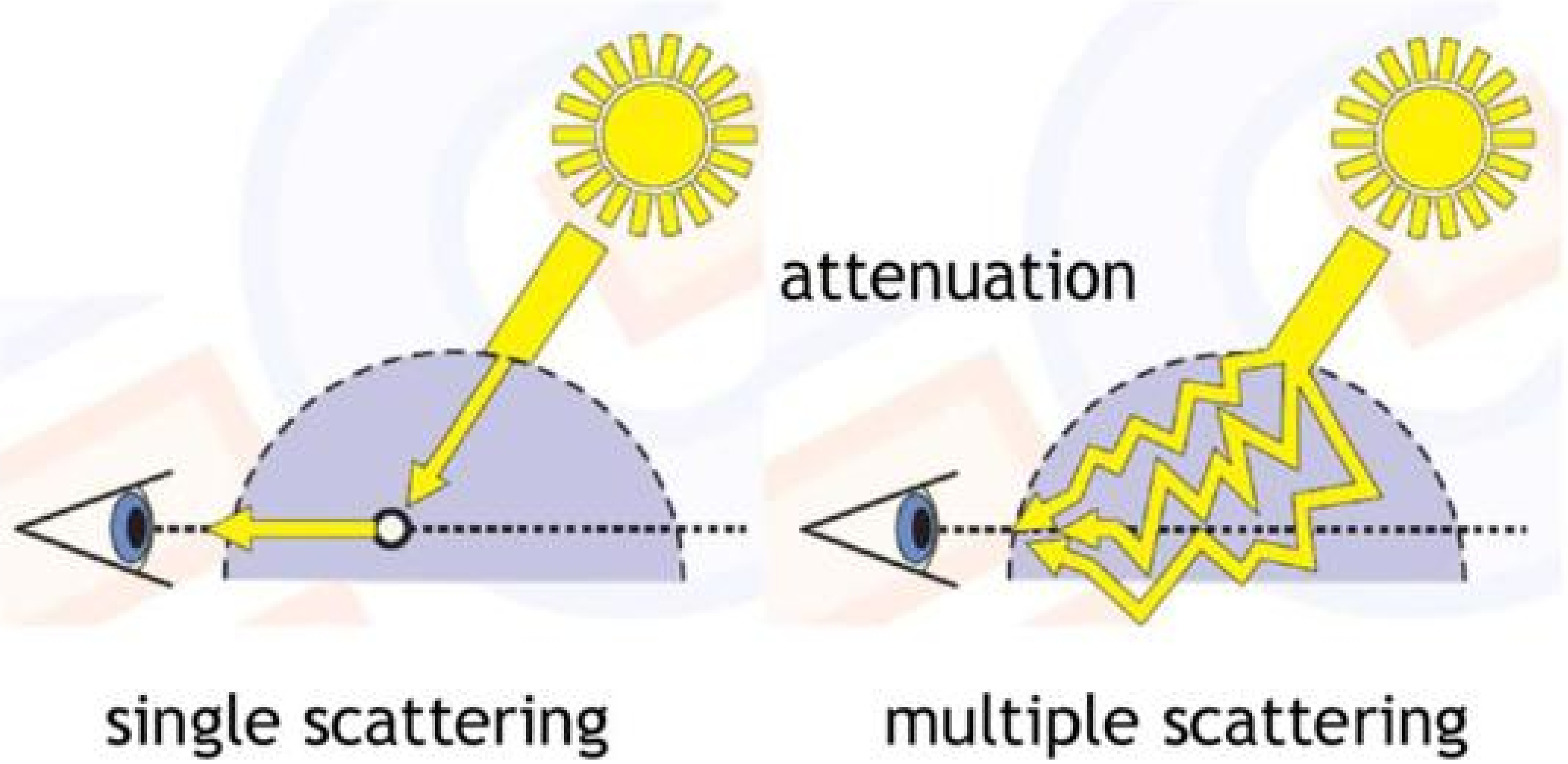
  Well-known for absorbing red light

Blue sky near zenith on sunset



Neptune covered by CH4

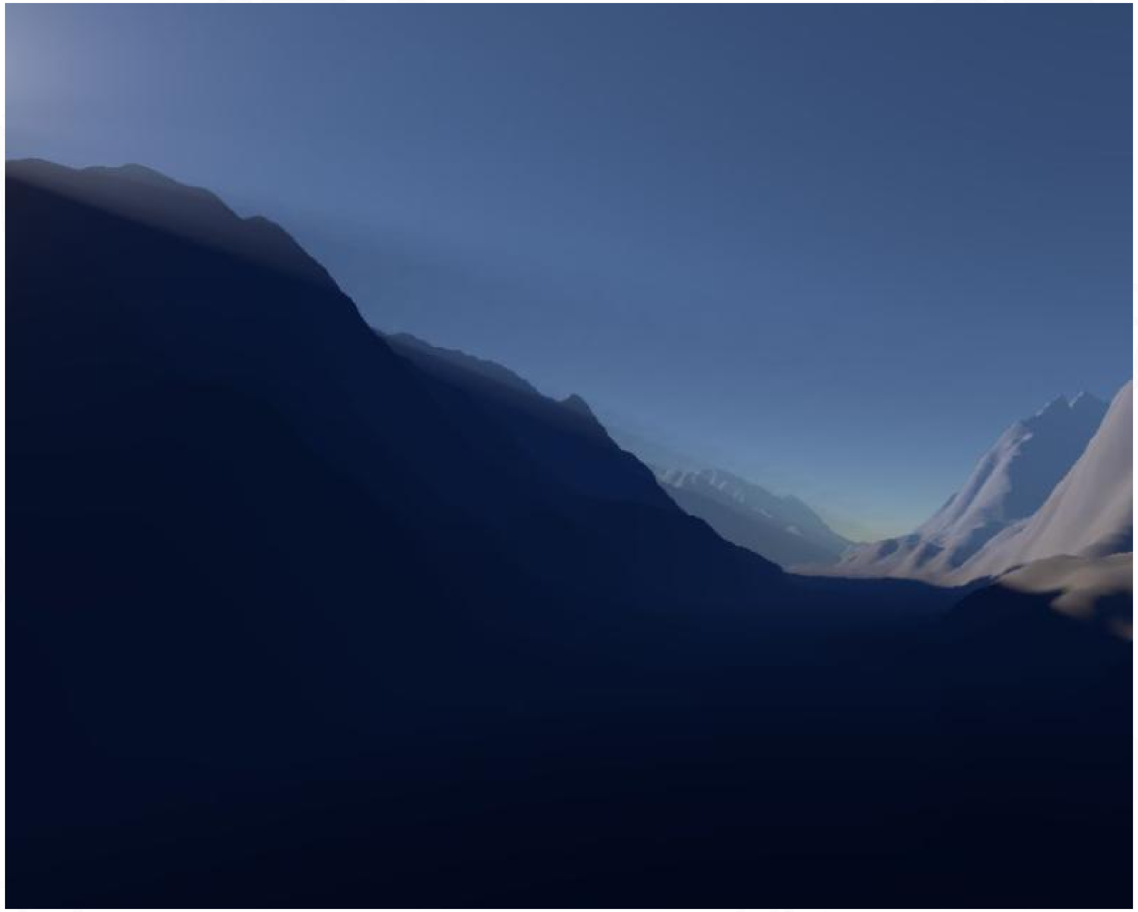
Single Scattering vs. Multi Scattering



$$L_1 = \int_A^B L_{P->A} ds \qquad L_{n+1} = \int_A^B \int_{4\pi} L_n(p,v') \cdot S(\lambda,\theta,\mathrm{h}) \cdot T(p -> A) dv' ds$$

Single Scattering vs. Multi Scattering

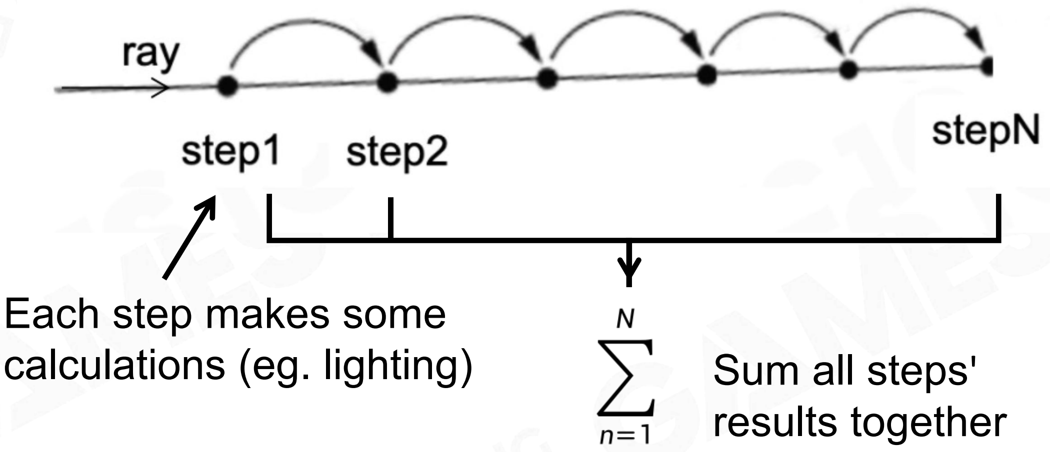
Single Scattering

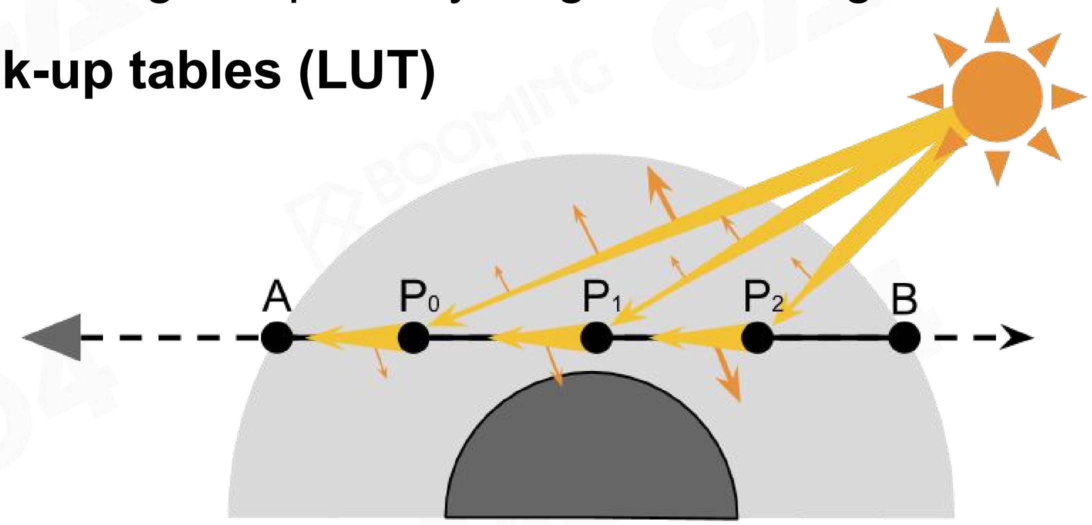


Multi Scattering

Ray Marching

- **Ray marching** is a popular method to integrate function along a path
- We use ray marching to calculate final radiance for a given point by single scattering
- The integrated radiance is usually stored in **look-up tables (LUT)**

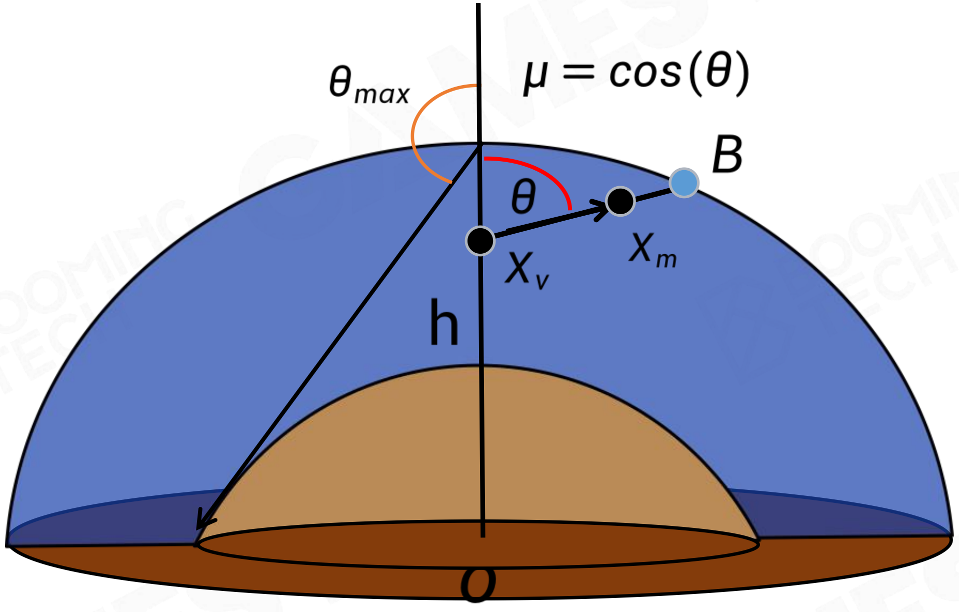




$$L_{sun}\int_A^B S(\lambda,\theta,\mathrm{h})\cdot(T(sun->P)+T(P->A))ds$$

Single Scattering Integration

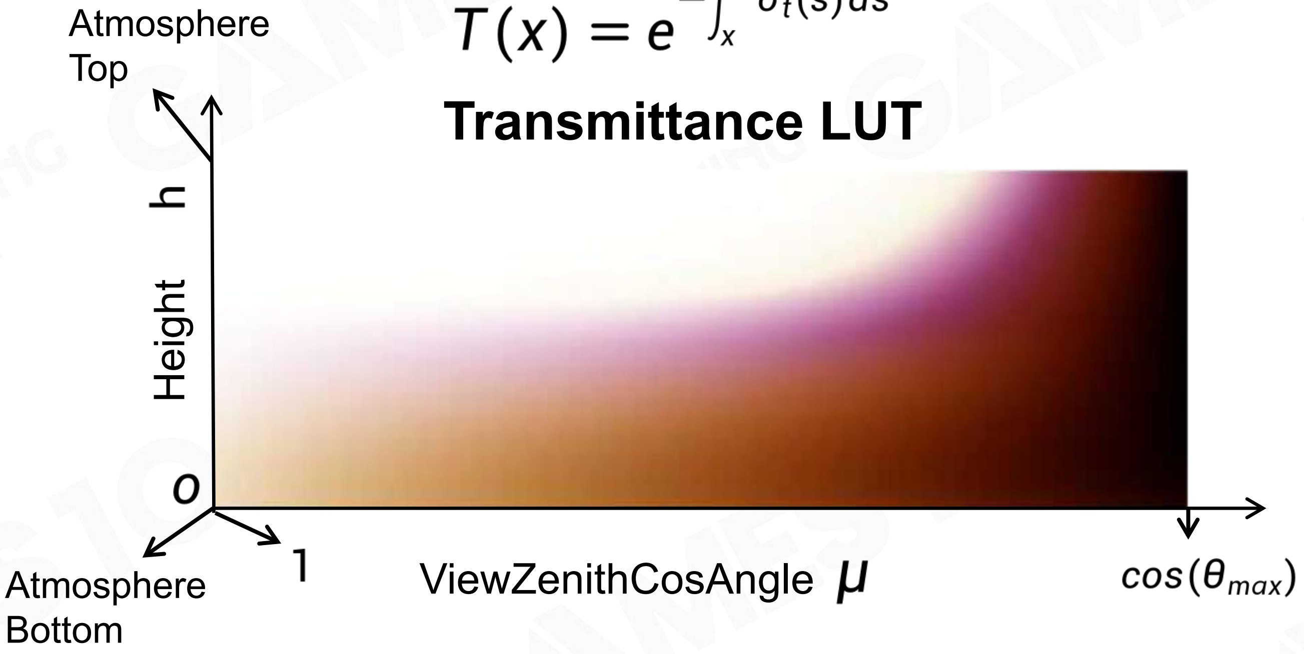
Precomputed Atmospheric Scattering



$$T(X_v -> X_m) = \frac{T(X_v -> B)}{T(X_m -> B)}$$

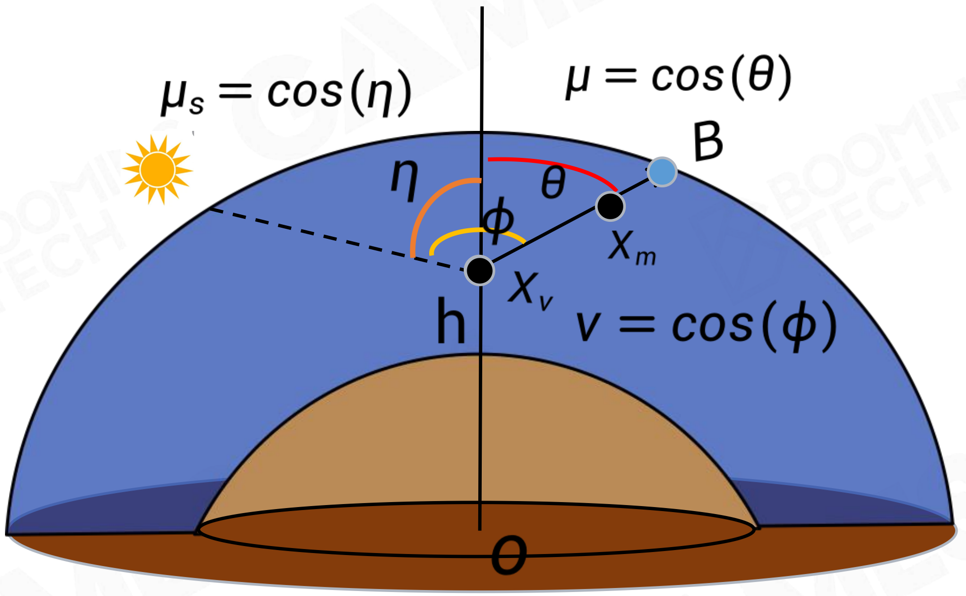
$$T(x) = e^{-\int_x^P \sigma_t(s)ds}$$

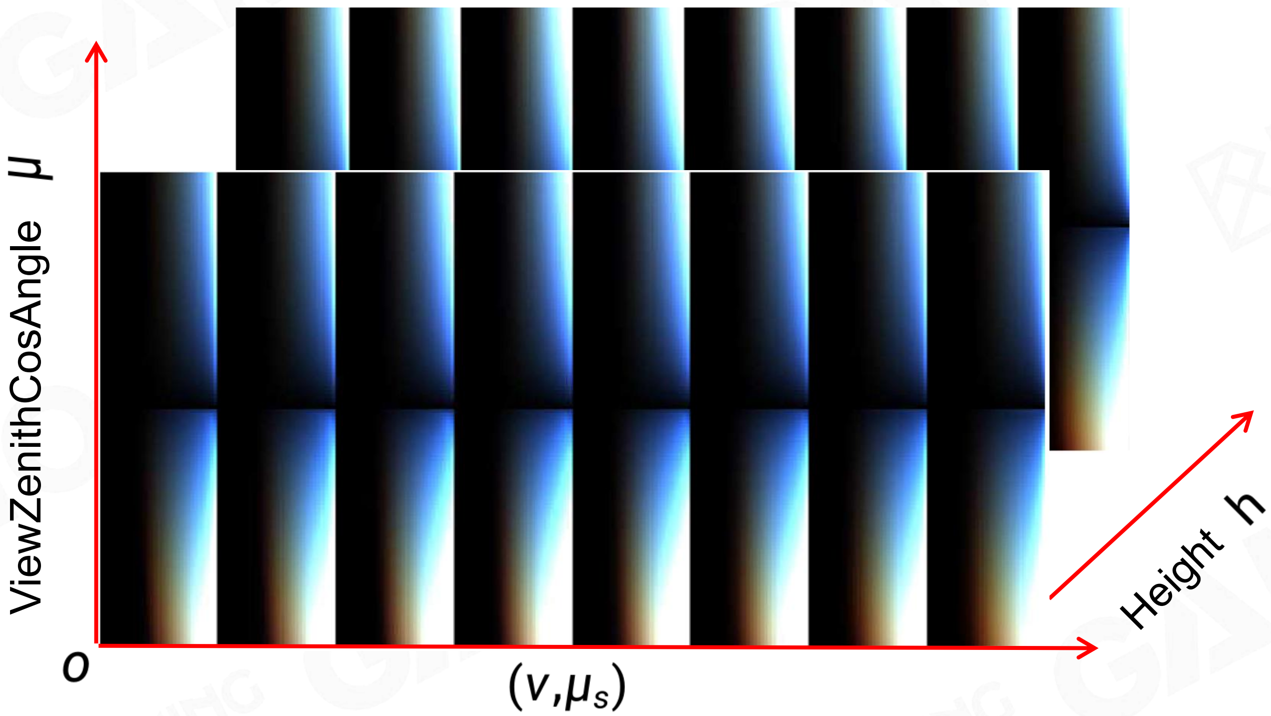
Transmittance LUT



https://ebruneton.github.io/precomputed_atmospheric_scattering/

Precomputed Atmospheric Scattering





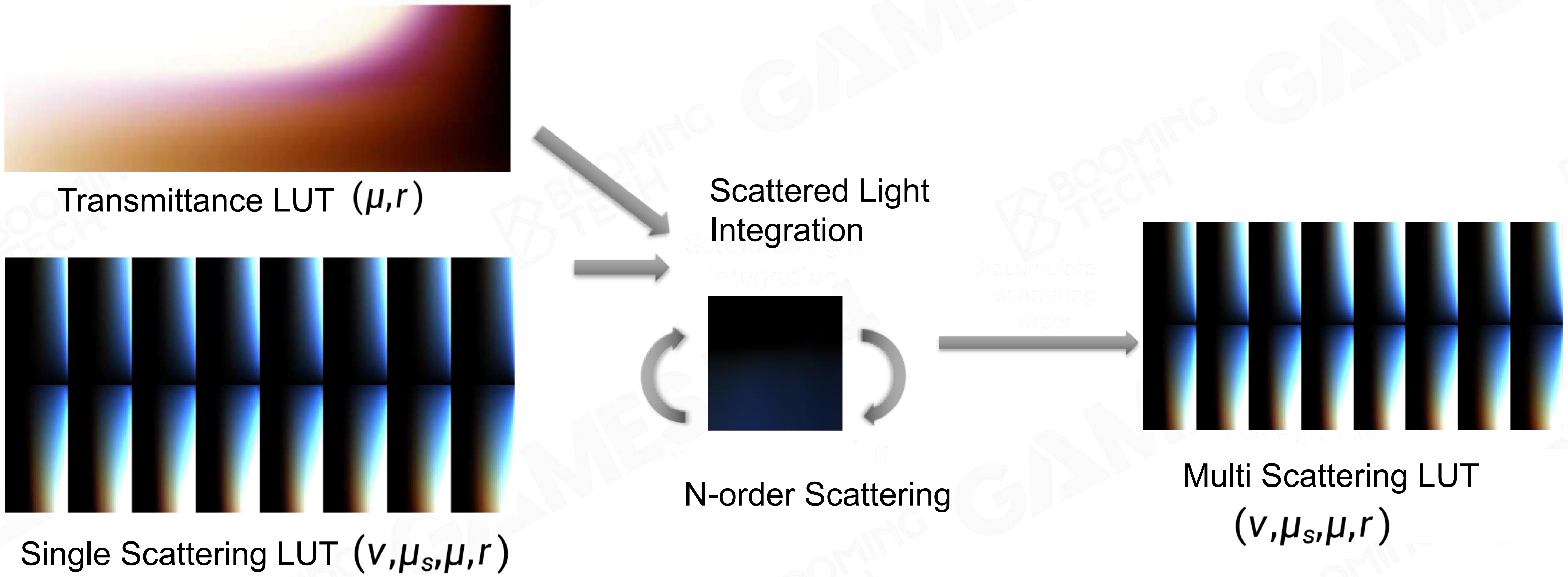
$$L_{sun}\int_A^B S(\lambda,\theta,h)\cdot(T(sun \rightarrow P)+T(P \rightarrow A))ds$$

ViewSunCosAngle x 8, SunZenithCosAngle x 32

$$L(X_v \rightarrow X_m) = L(X_v \rightarrow B) - L(X_m \rightarrow B)\cdot T(X_v \rightarrow X_m)$$

Precomputed Atmospheric Scattering

Multi Scattering LUT






Precomputed Atmospheric Scattering

Challenges of Precomputed Atmospheric Scattering

- Precomputation Cost
  - Multi-scattering iterations are very expensive
  - Hard to generate atmosphere LUT on low-end devices (ie. mobile)
- Authoring and Dynamic Adjustment of Environments
  - Artist can't change scattering coefficients on the fly
  - Hard to render effects like weather from sunny to rain fog, space travel among planets
- Runtime Rendering Cost
  - Expensive per-pixel multi high dimensional texture sampling for transmittance LUT and multi scattering LUT (always need to down-sample for efficiency)

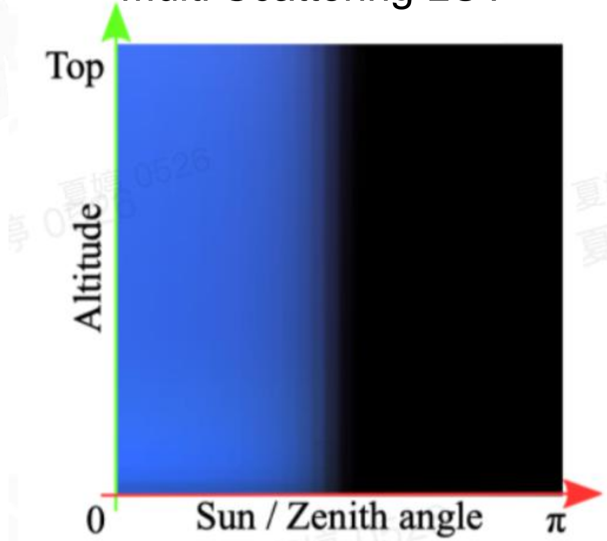
A Scalable and Production Ready Sky and Atmosphere Rendering Technique
https://diglib.eg.org/bitstream/handle/10.1111/cgf14050/v39i4pp013-022.pdf

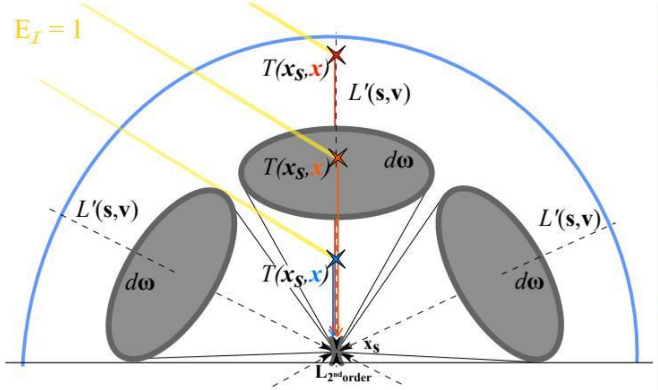
Production Friendly Quick Sky and Atmosphere Rendering

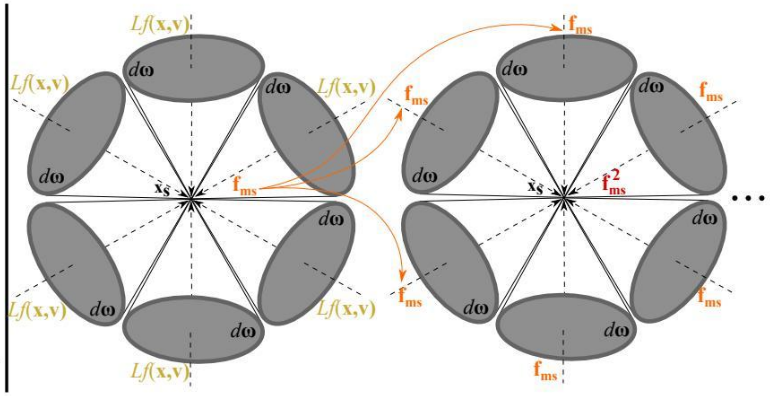
Simplify Multi-scattering Assumption

- Scattering events with order greater or equal to 2 are executed using an **isotropic phase function**
- All points within the neighborhood of the position we currently shade **receive the same amount of second order scattered light**
- **Visibility is ignored**

Multi Scattering LUT







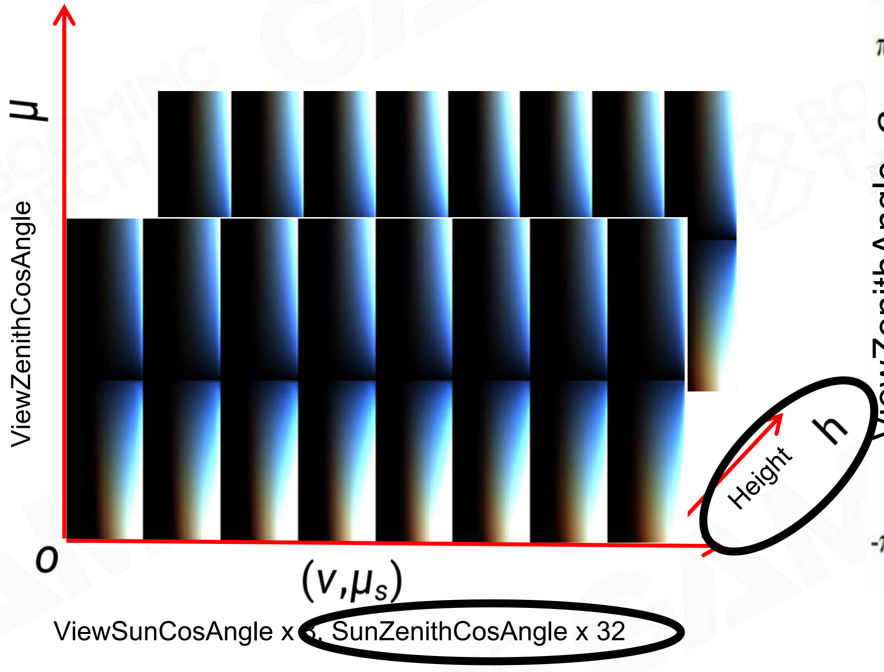
$$G_{n+1} = G_n * f_{ms}$$

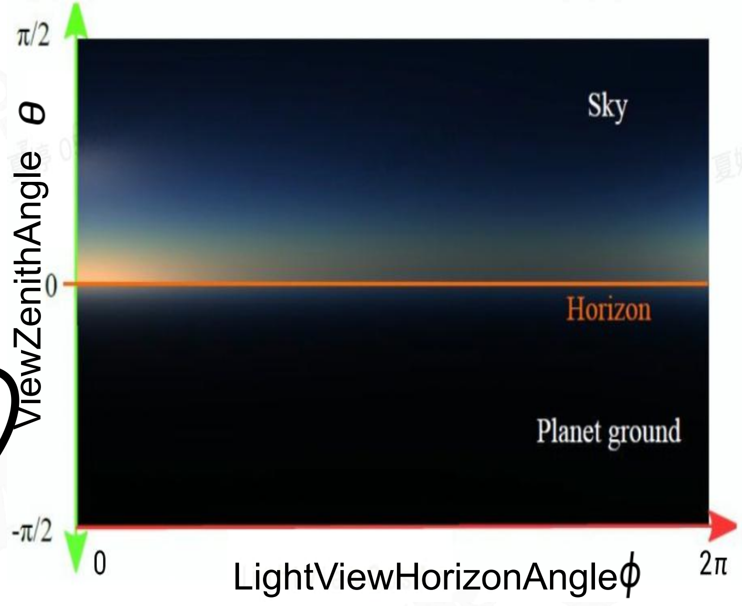
$$F_{ms} = 1 + f_{ms} + f_{ms}^2 + f_{ms}^3 + \dots = \frac{1}{1 - f_{ms}}$$

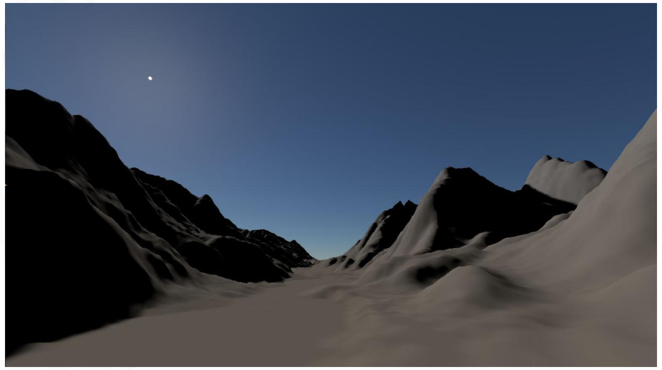
$$\Psi_{ms} = L_{2^{nd}order} F_{ms}$$

Production Friendly Quick Sky and Atmosphere Rendering

Fixed view position and sun position to remove 2 dimensions out of LUT

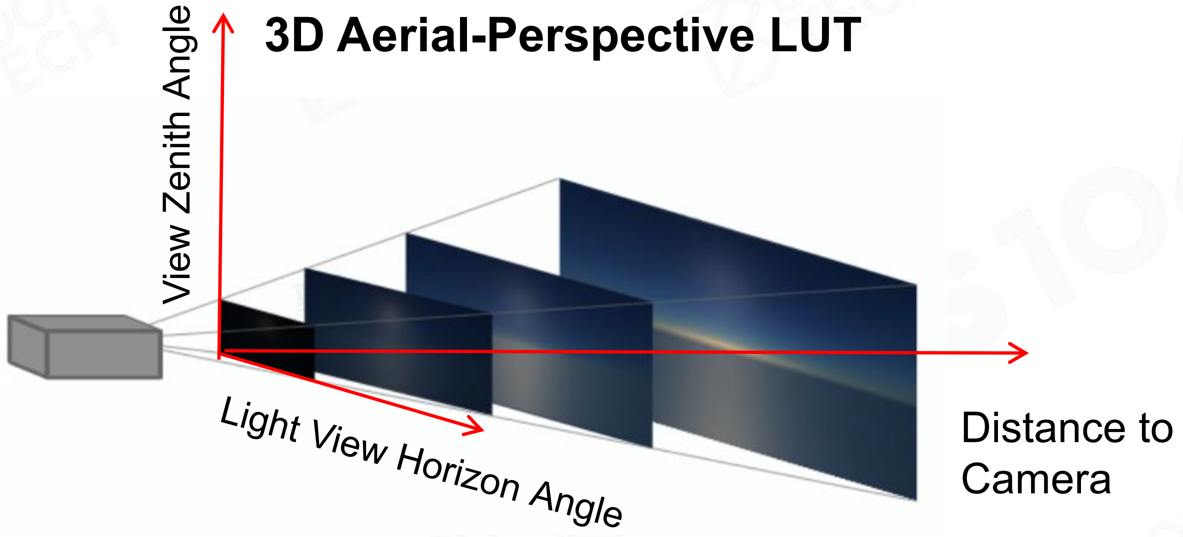


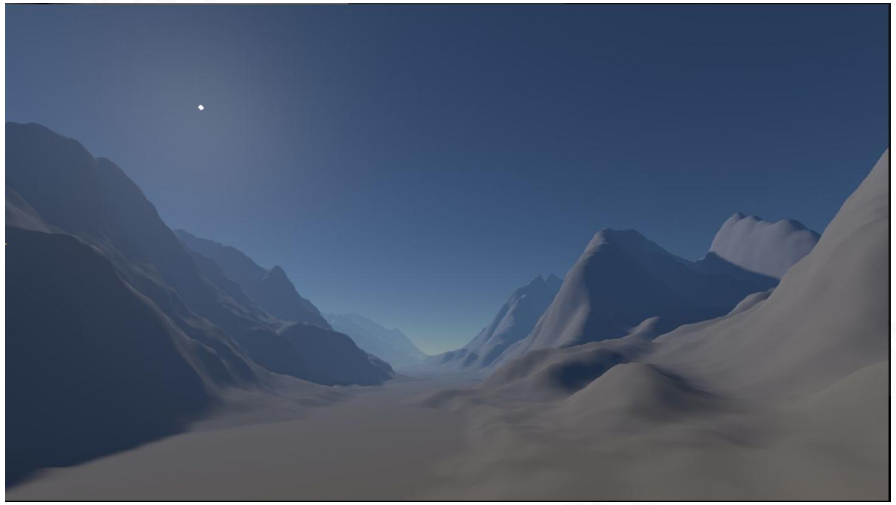




Production Friendly Quick Sky and Atmosphere Rendering

- Generated a 3D LUT to evaluate aerial-perspective effects by ray marching



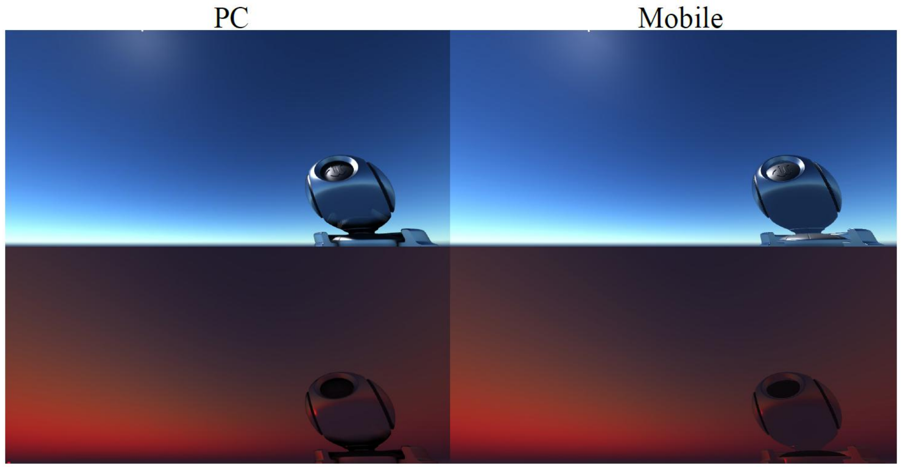


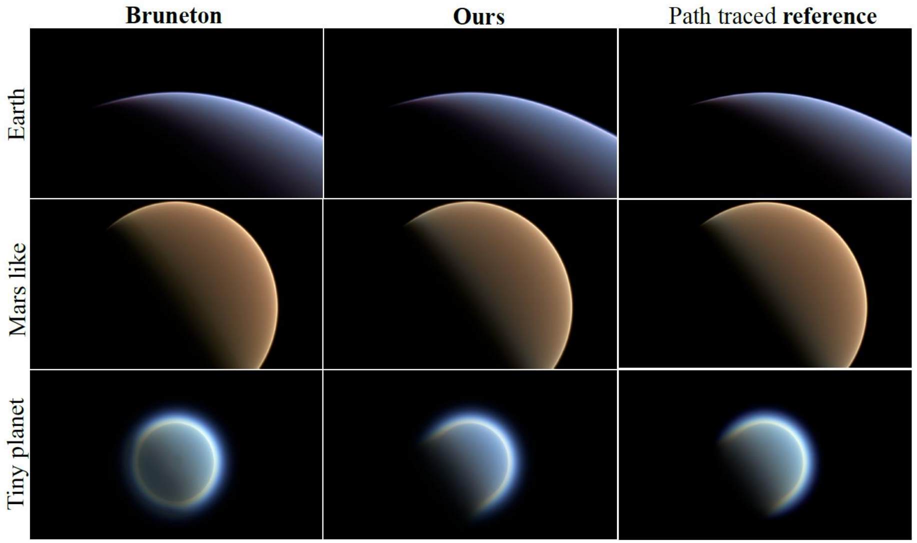
Good Balance of Performance and Effect

- **Scalable from mobile to high-end PCs**

PC			
LUT	Resolution	Step count	Render time
Transmittance	256×64	40	0.01ms
Sky-View	200×100	30	0.05ms
Aerial perspective	32^3	30	0.04ms
Multi-scattering	32^2	20	0.07ms
Mobile (iPhone 6s)			
LUT	Resolution	Step count	Render time
Transmittance	256×64	40	0.53ms
Sky-View	96×50	8	0.27ms
Aerial perspective	$32^2 \times 16$	8	0.11ms
Multi-scattering	32^2	20	0.12ms

Performance for each step of method, as measured on PC (NV 1080) and a mobile device (iPhone 6s)



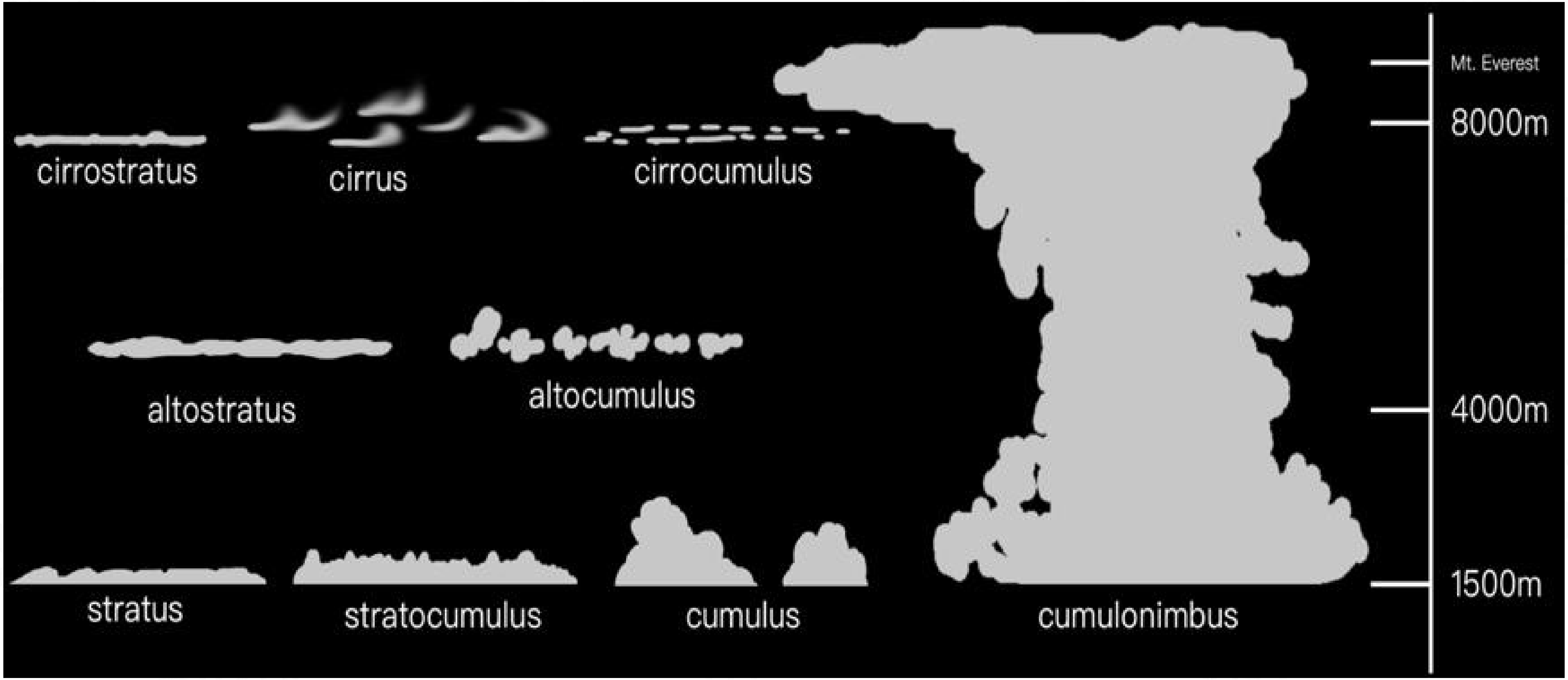


Video of Atmosphere Demo

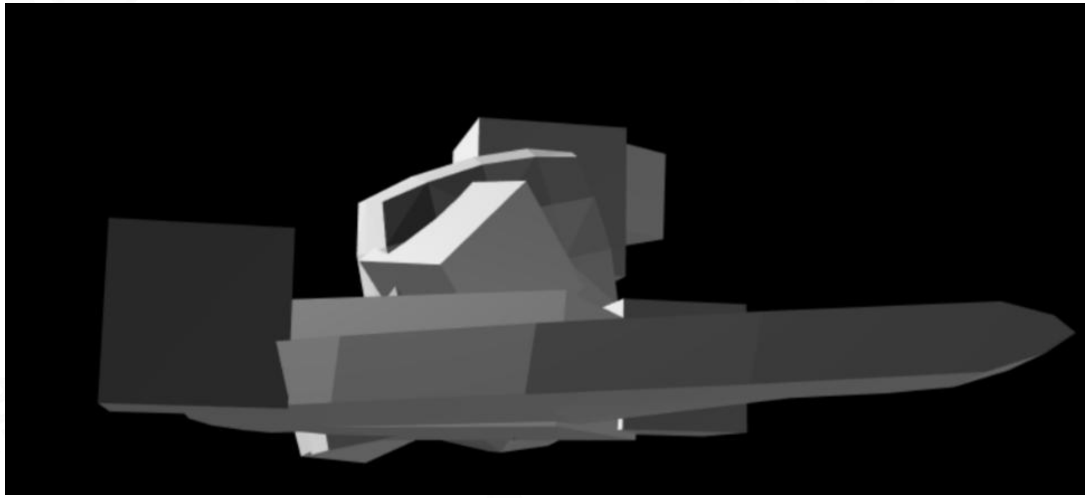
"Paint" Cloud

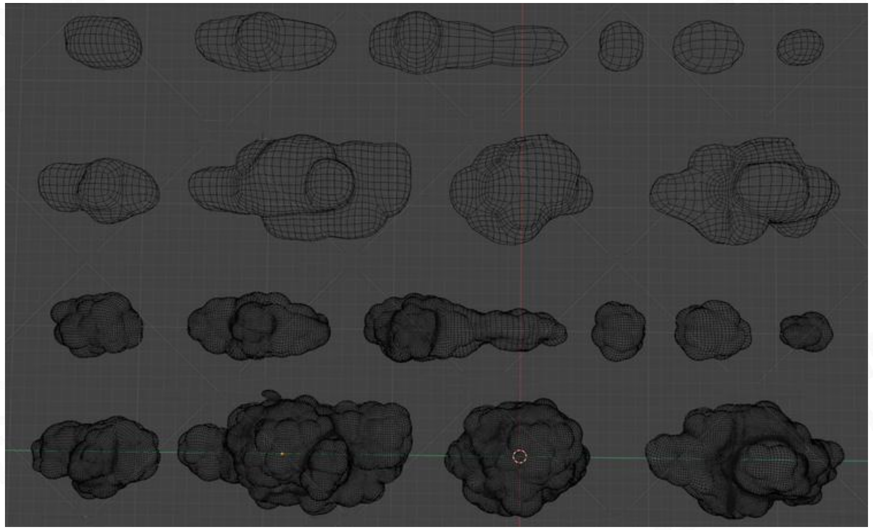
Luke Howard, Meteorologist, 1802

Cloud Type


cirrostratus
cirrus
cirrocumulus
altostratus
altocumulus
stratus
stratocumulus
cumulus
cumulonimbus
Mt. Everest
8000m
4000m
1500m

Mesh-Based Cloud Modeling





Pros
- High quality

Cons
- Overall expensive
- Do not support dynamic weather

Billboard Cloud

Pros
- Efficient

Cons
- Limited visual effect
- Limited cloud type

Volumetric Cloud Modeling

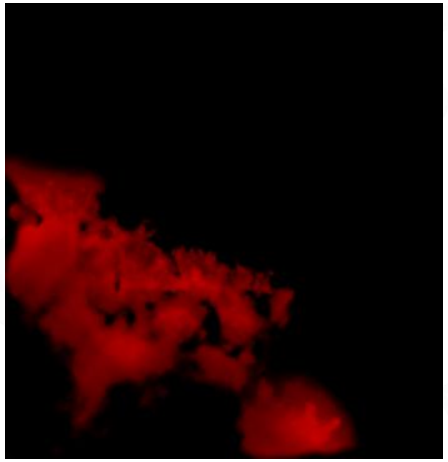
Pros

- Realistic cloud shapes
- Large scale clouds possible
- Dynamic weather supported
- Dynamic volumetric lighting and shadowing

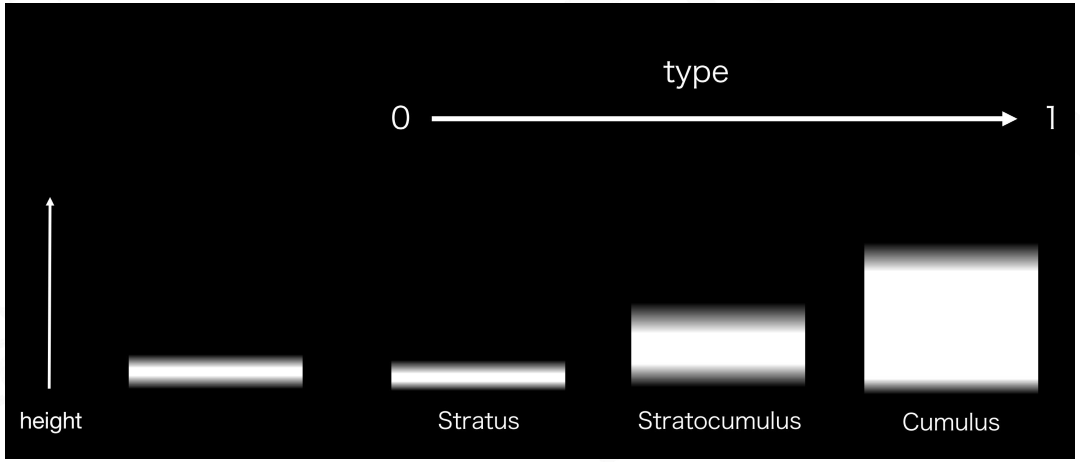
Cons

- Efficiency must be considered

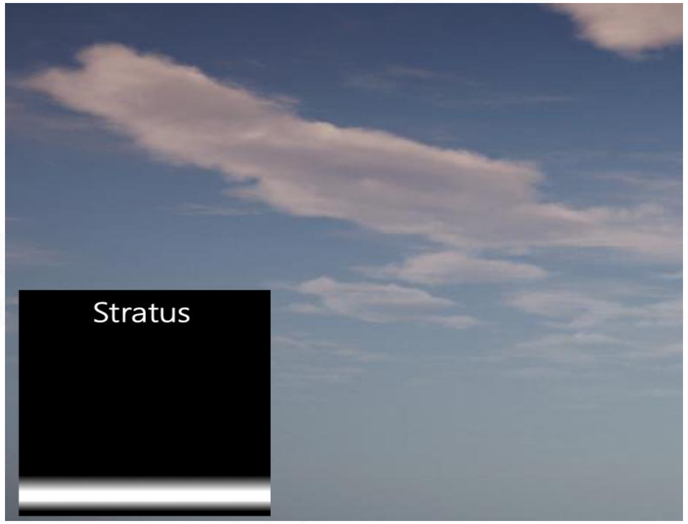
Weather Texture

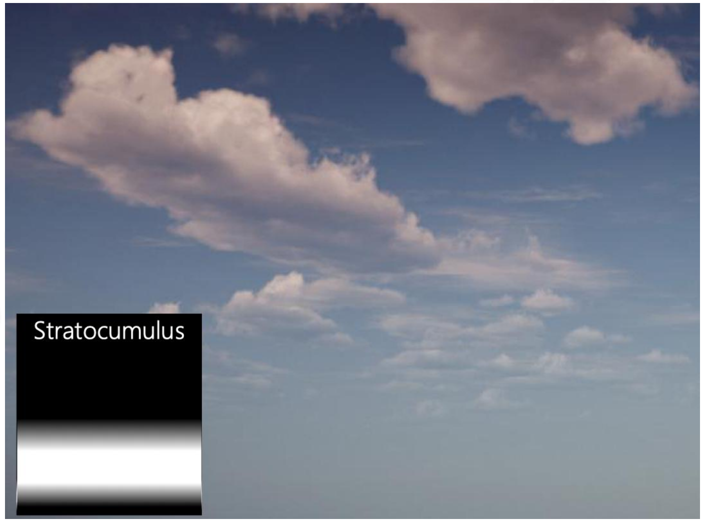


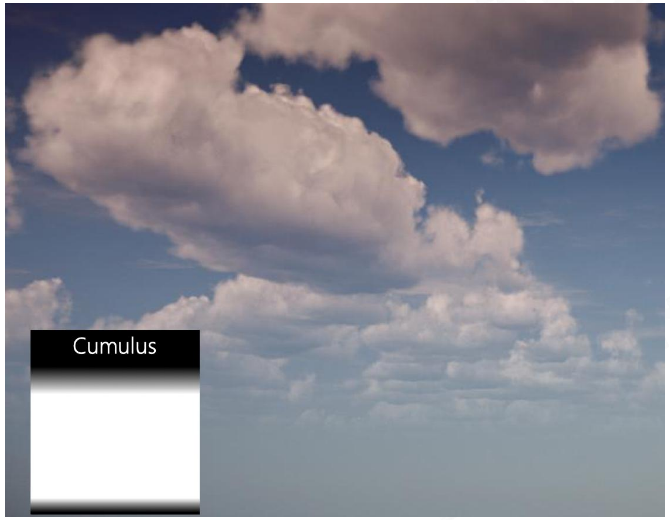
+



=

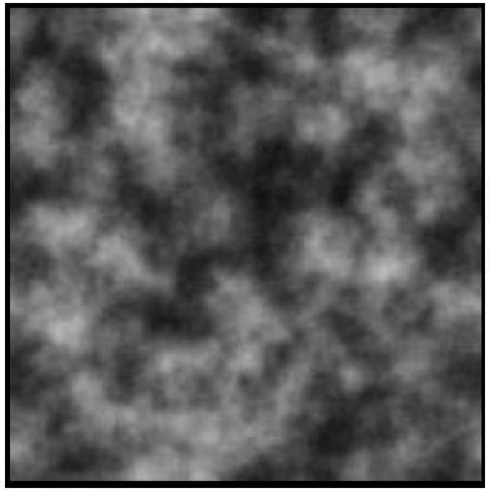


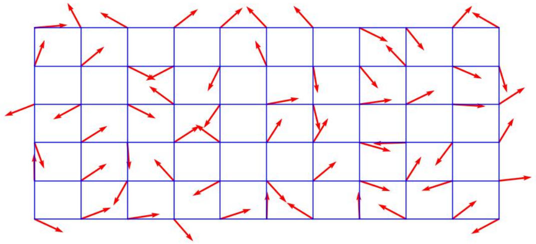




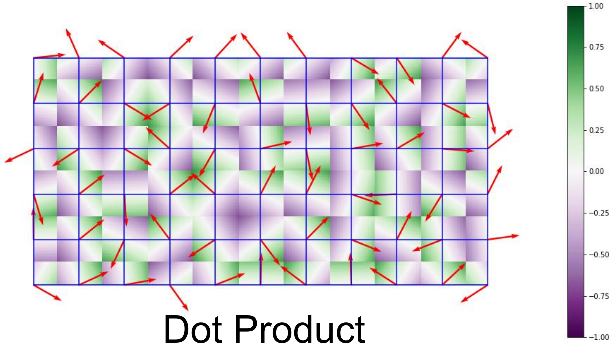
Noise Functions

Perlin Noise

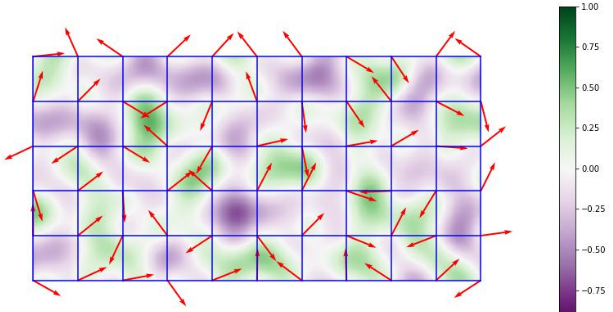




Grid Definition



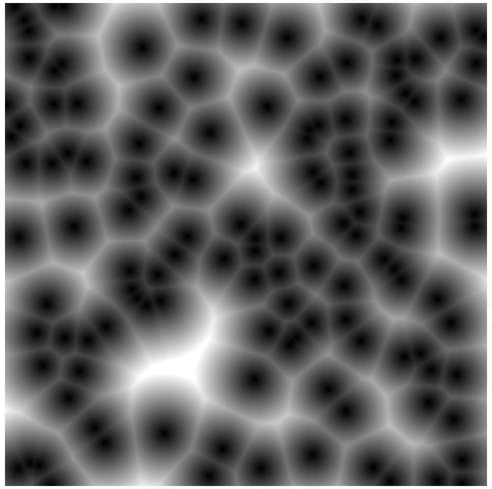
Dot Product



Interpolation

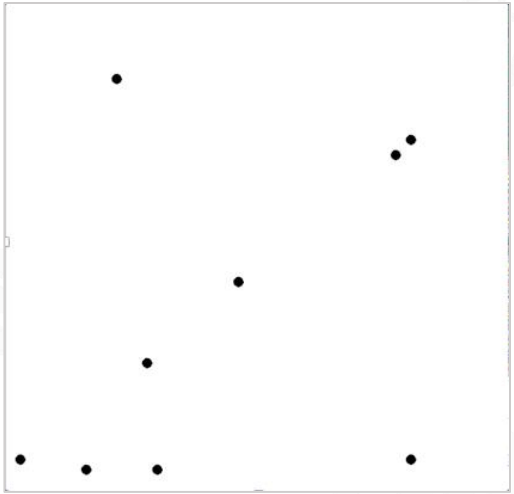
https://en.wikipedia.org/wiki/Perlin_noise

Worley Noise



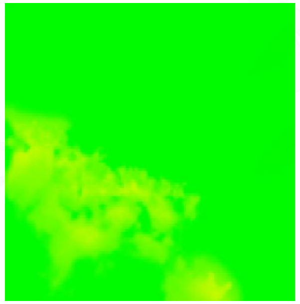
https://thebookofshaders.com/12/

Voronoi



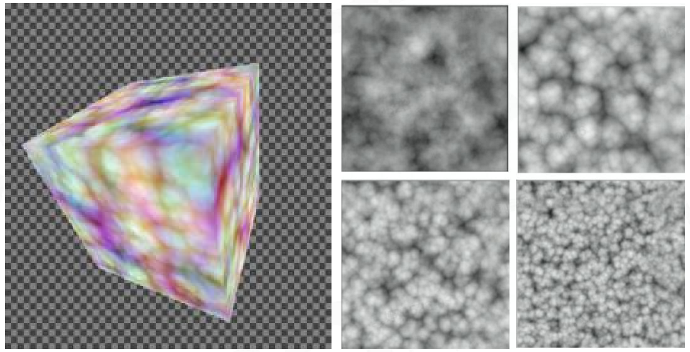
Cloud Density Model

Basic Distribution



Basic Shape

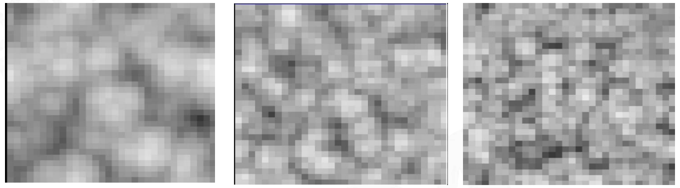
RGBA channels
R Perlin-Worley
GBA layered Worley

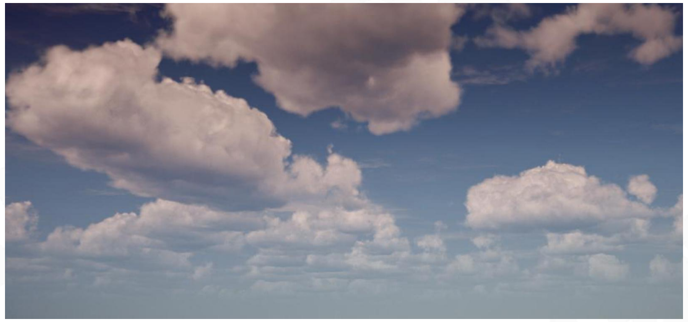


$$SN_{sample} = R(sn_r,\ (sn_g \times 0.625 + sn_b \times 0.25 + sn_a \times 0.125) - 1,\ 1,\ 0,\ 1)$$

More Details

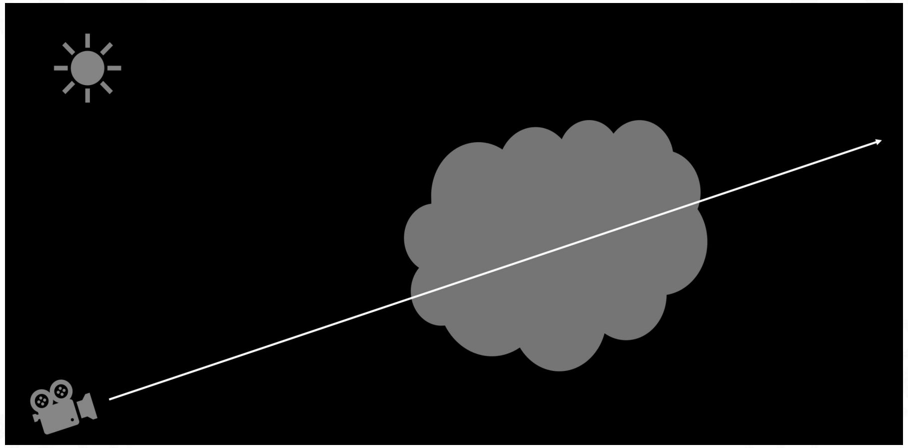
3 low resolution Worley



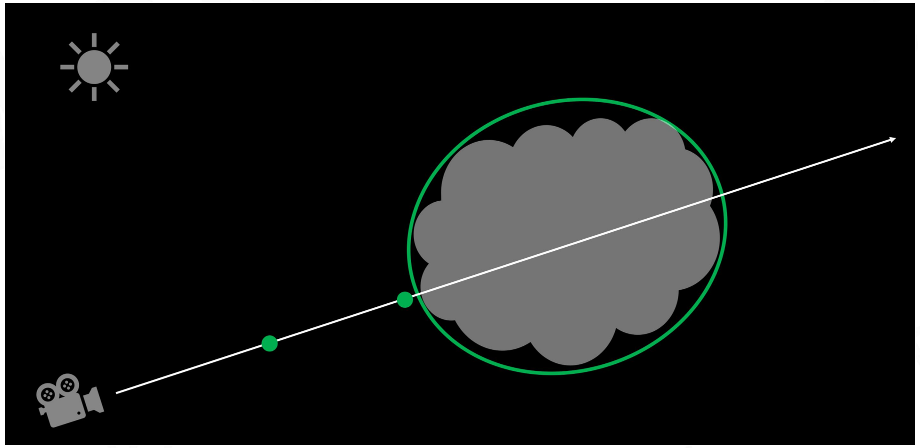


$$DN_{fbm} = dn_r \times 0.625 + dn_g \times 0.25 + dn_b \times 0.125$$

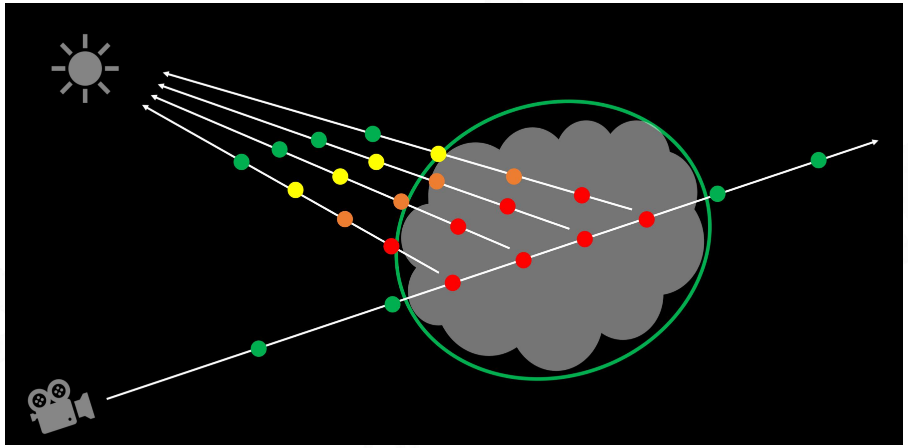
Rendering Cloud by Ray Marching



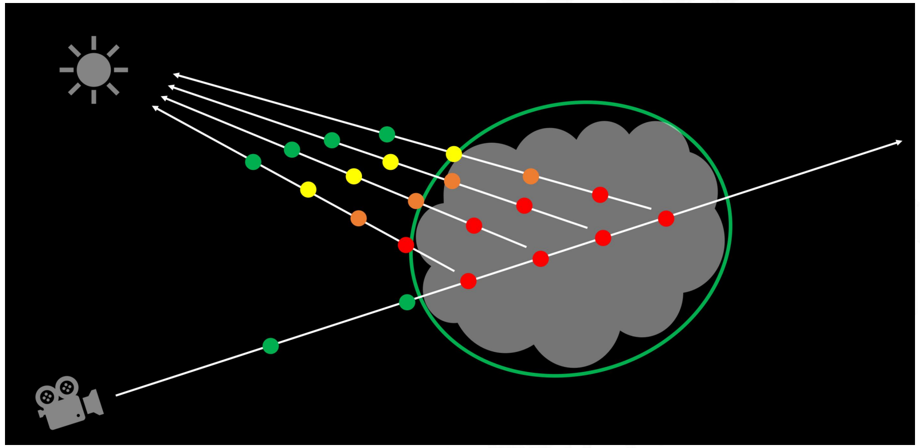
Step 1 : Cast ray for each screen pixel



Step 2 : Big step marching until hitting cloud



Step 3 : Dense step sampling inside cloud



Step 4 : Gather radiance scattered from sun


Video of Volume Cloud

Pilot Engine V0.0.2 Released - 12 April

Bugfixes

- Fixed the transform of rigid bodies of objects
- Fixed crashes when reloading current level
- Fixed transforming objects by dragging axes of Transform – Component in Component Details Panel
- Fixed specular calculation when roughness is 0
- Fixed compilation and crashes on M1 macOS

Optimizations

- Optimized the display performance of the file tree in File Content Panel
- Optimized the coloring of axes of Transform Component in Component Details Panel
- Prefer independent graphics card when initializing Vulkan

Contributors

ShenMian, KSkun, and 19 other contributors

Optimization of Course Arrangement

Take 1 week break every 3 lectures from lecture08

- The course team needs a break to better prepare for the course
- Leave more time for students to digest knowledge and catch up with homework

Q&A

Lecture 06 Contributor

- 一将
- 光哥
- 炯哥
- 玉林
- 小老弟
- 建辉
- 爵爷
- Jason
- 砚书
- BOOK
- MANDY
- 俗哥
- 金大壮
- Leon
- 梨叔
- Shine
- 邓导
- Judy
- QIUU
- C佬
- 阿乐
- 阿熊
- CC
- 大喷

Enjoy ;) Coding

Course Wechat

Follow us for further information