N-BVH: Neural ray queries with bounding volume hierarchies

Authors: Philippe Weier, Alexander Rath, Élie Michel, Iliyan Georgiev, Philipp Slusallek, Tamy Boubekeur

Presenter: Yansong Yu

Date: 2024/07/02

Background

- Conventional rendering techs
  - Ray tracing
  - Prefiltering
  - Bounding volume hierarchy (BVH)
- Neural rendering techs
  - NeRF (Neural Radiance Field) (2020)
  - Neural Radiosity (2021)
  - Instant-ngp (2022)

Ray tracing

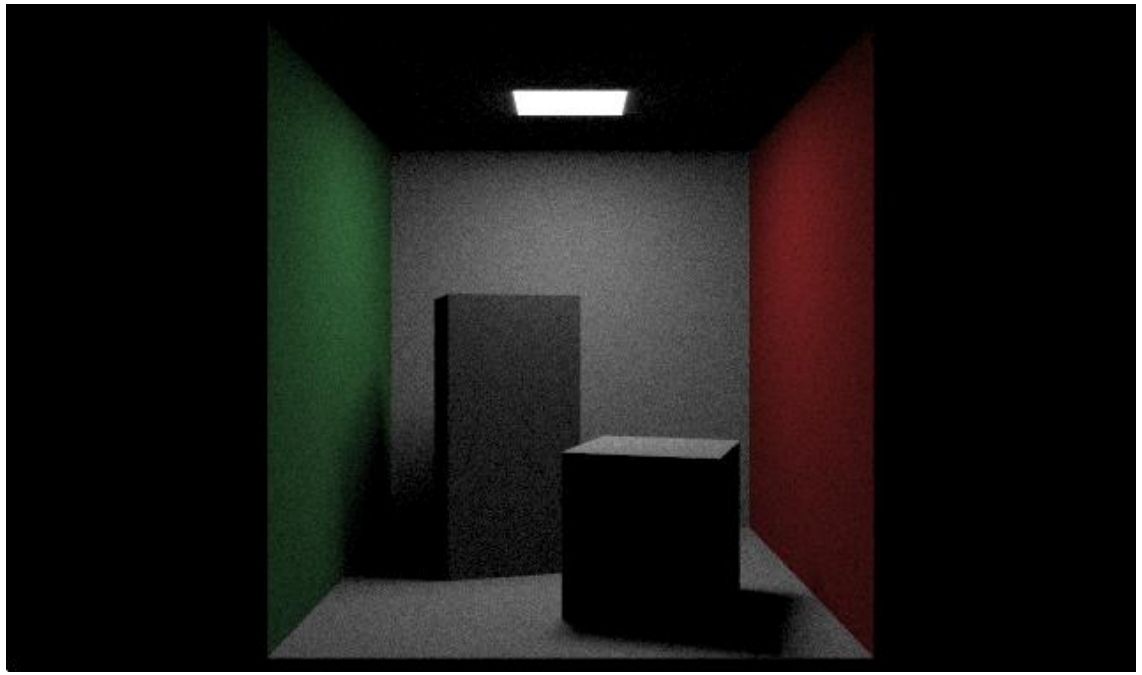
- Like rasterization, ray tracing is a technique for rendering a 3D scene to a 2D image.
- Ray tracing methods can be divided into simple ones and advanced ones.

Ray tracing

- For example, one of the simplest ray tracing method in 3D cases can be described like:
  - Shoot a ray ***R*** from the camera;
  - Find the ***closest*** intersection of ***R*** and ***every primitive*** in the scene;
  - If an intersection point ***P*** is found in the last step, go to next step;
  - Shoot a shadow ray ***S*** from ***P*** to the light to test if ***P*** is shaded by ***any*** obstacles
  - If ***P*** is shaded, then color the pixel to black, otherwise, use the surface properties (albedo, normal) at ***P*** and light intensity (commonly, RGB) and direction to color the pixel;
  - Go back to the first step.

Ray tracing

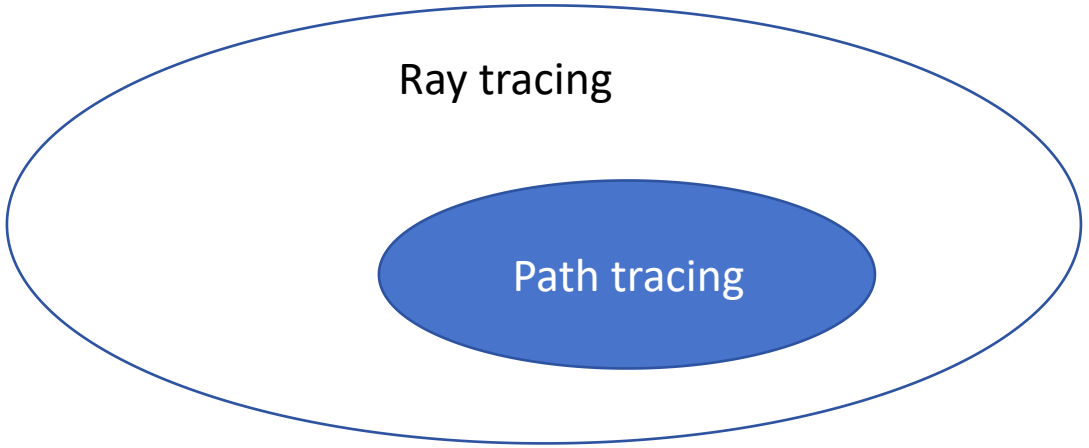
- From the simplest ray tracing method just mentioned, we can generate the following images. (The effect is just like simple rasterization)



https://www.shadertoy.com/view/XlGcWD

Ray tracing

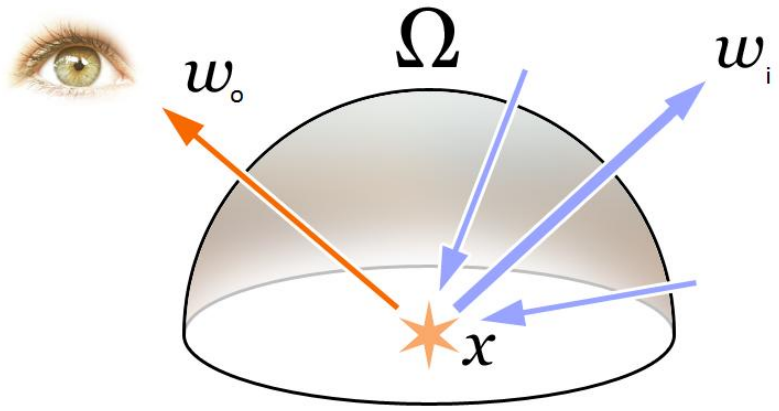
- One example from advanced ray tracing methods is called path tracing.
- The relationship between path tracing and ray tracing is shown below.



Ray tracing

- Path tracing is guided by ***the rendering equation*** (James T. Kajiya. 1986.).
- Rendering equation generalize the realistic rendering problem to an **energy** equation.

$$L_r(x,\omega_o) = E(x,\omega_o) + \int_{\Omega} f_r(x,\omega_i,\omega_o)\cdot L_o(x',\omega_i)\cdot(\omega_i\cdot n)d\omega_i$$



Wikipedia Contributors, "Rendering equation," Wikipedia, Sep. 22, 2019. https://en.wikipedia.org/wiki/Rendering_equation

Ray tracing

- Here are some BRDF, $F(\omega_o, \omega_i)$ examples:

$$albedo/\pi$$

Lambertian (Diffuse BRDF)

$$\frac{albedo}{\pi}(A + B\max(0, \cos(\phi_{wo} - \phi_{wi}))\sin\alpha\tan\beta)$$

Oren Nayar (Diffuse BRDF)

PBRT 3ed 2018 ch 8.4.1

$$\frac{1+s}{2\pi}spec * (R_i \cdot \omega_o)^s$$

Phong (Specular BRDF)

$$\frac{FGD}{4(n \cdot \omega_i)(n \cdot \omega_o)}$$

GGX (Specular BRDF)

$$\begin{aligned} f_{\text{disney}} = &(1 - \text{specularTransmission}) \cdot (1 - \text{metallic}) \cdot f_{\text{diffuse}} + \\ &(1 - \text{metallic}) \cdot \text{sheen} \cdot f_{\text{sheen}} + \\ &(1 - \text{specularTransmission} \cdot (1 - \text{metallic})) \cdot \hat{f}_{\text{metal}} + \\ &0.25 \cdot \text{clearcoat} \cdot f_{\text{clearcoat}} + \\ &(1 - \text{metallic}) \cdot \text{specularTransmission} \cdot f_{\text{glass}} \end{aligned}$$

Disney BSDF
(All in one BSDF)
https://cseweb.ucsd.edu/~tzli/cse272/wi2023/homework1.pdf

Ray tracing

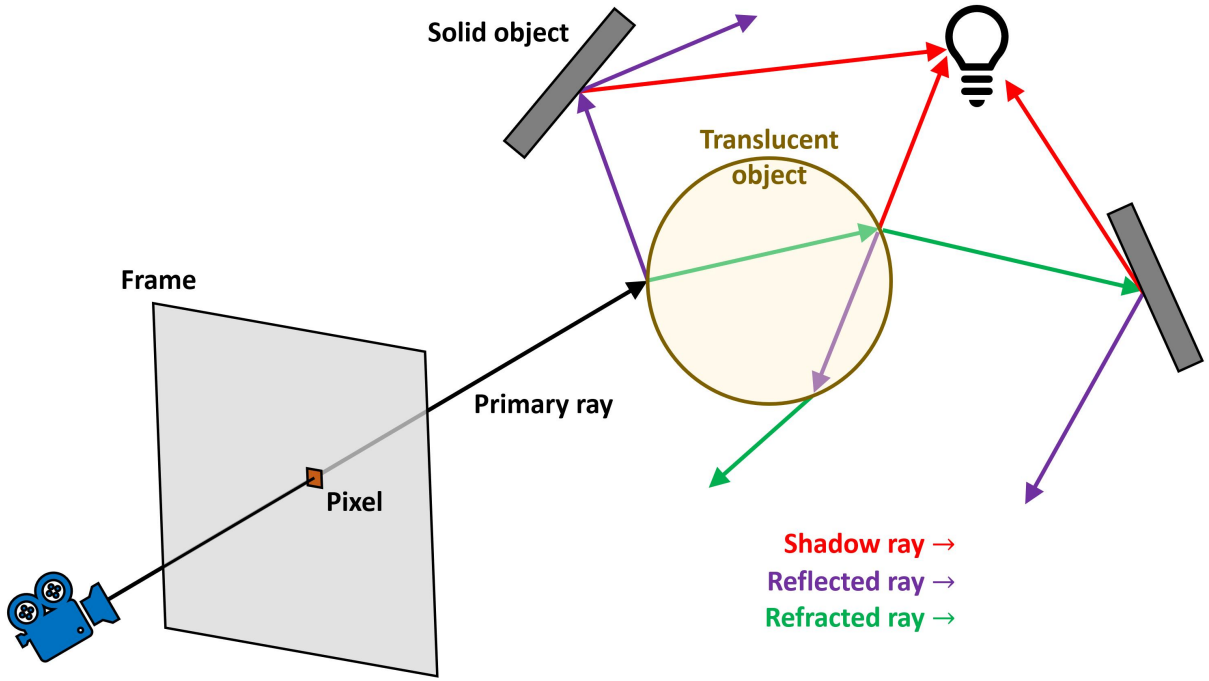
- Take a look again on the ***rendering equation***, it has integral and the integral is **not closed**, so it **can not** be solved **analytically and directly**.
- **Path tracing** is an unbiased way to solve this equation **numerically based on Monte Carlo Estimation (*PBRT 3ed 2018 Ch 14.5*)**.

$$L_o(x,\omega_o) = E(x,\omega_o) + \int_{\Omega} f_r(x,\omega_i,\omega_o) \cdot L_o(x',\omega_i) \cdot (\omega_i \cdot n) d\omega_i$$

- In path tracing, to calculate the color of a specific pixel, multiple ***path*s** needs to be traced throughout the scene.
- ***path*: end-to-end connected line segments start from camera/eye to the light source/background, each line segment is a bounced ray.**

Ray tracing

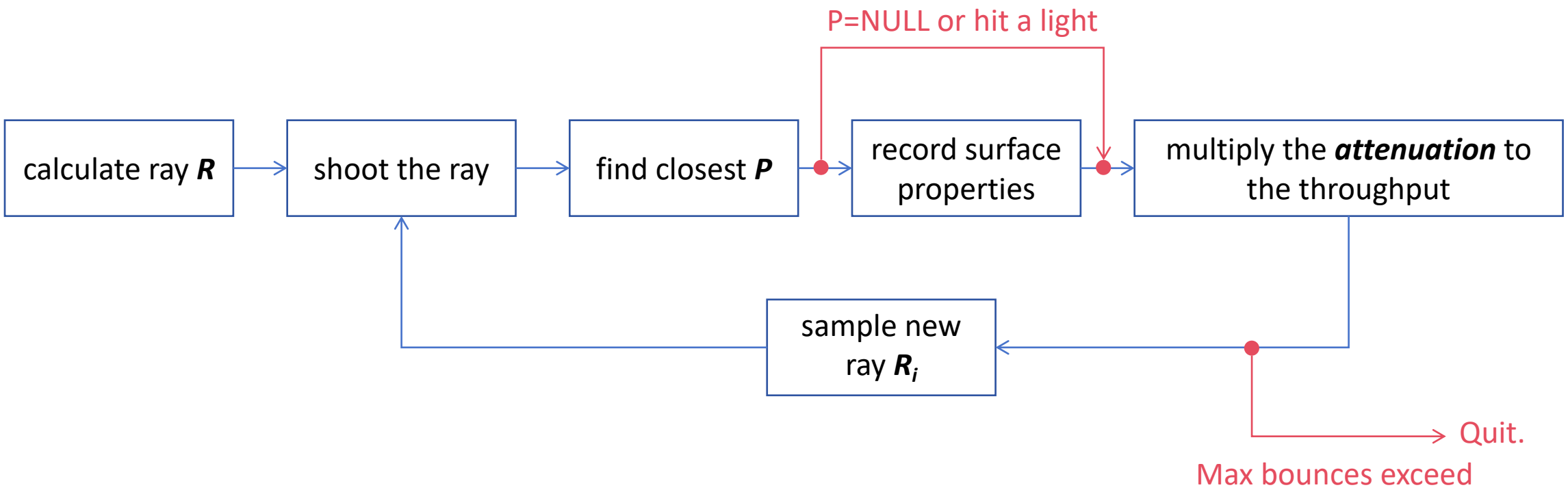
- The ***path*** in path tracing.



"Path Tracing vs Ray Tracing, expliqué," www.netcost-security.fr, Jul. 11, 2022. https://www.netcost-security.fr/actualites/105890/path-tracing-vs-ray-tracing-explique/ (accessed Jun. 26, 2024).

Ray tracing

- For a single ***path***, the routine:



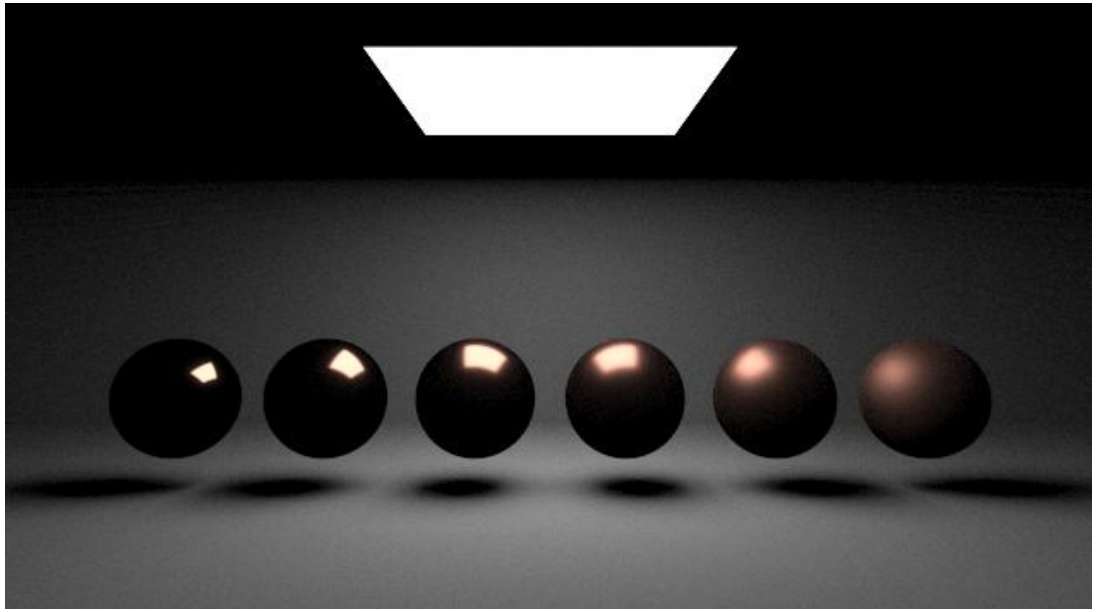
Ray tracing

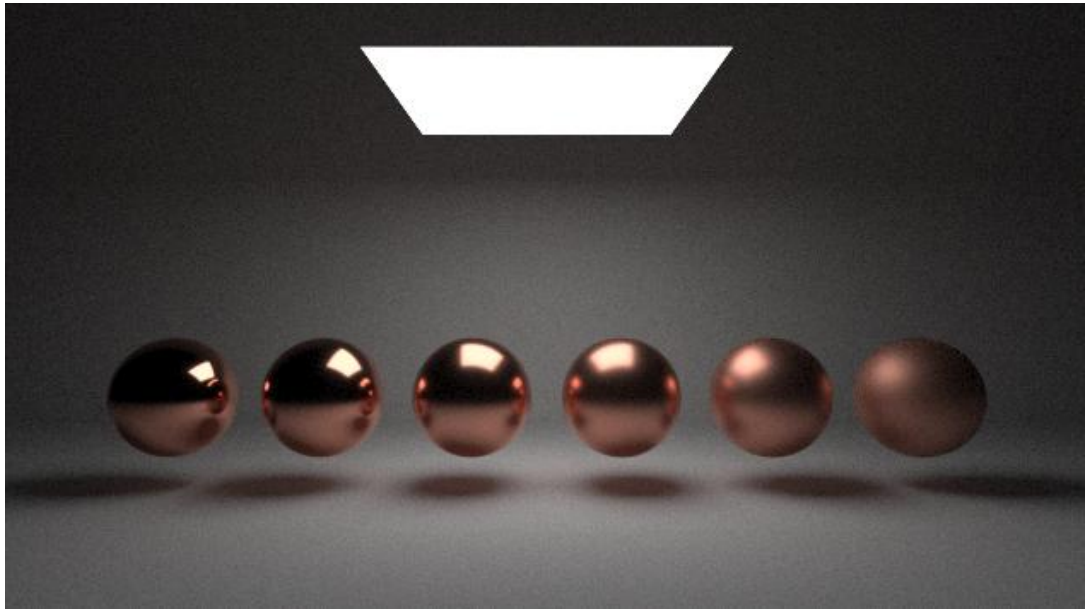
- Then, the color of $\boldsymbol{pixel_{i,j}}$:

$$color_{i,j} = \frac{1}{k} \sum_{k} color_{path_k}$$

Ray tracing

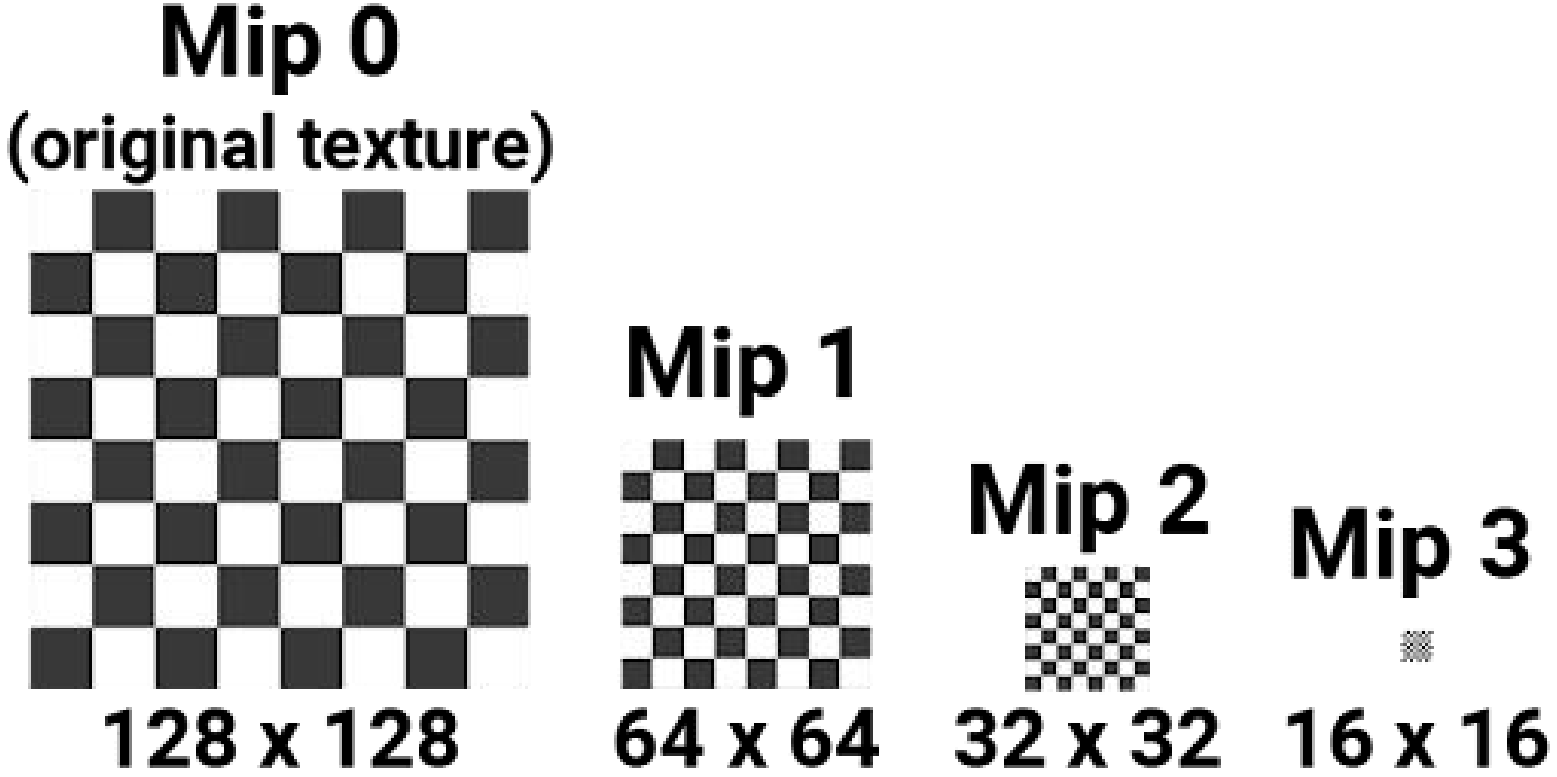
- Here is an intuitive comparison between path tracing and ray tracing in simplest case.





Prefilering

- From my understanding, “Mipmap” falls into this category.

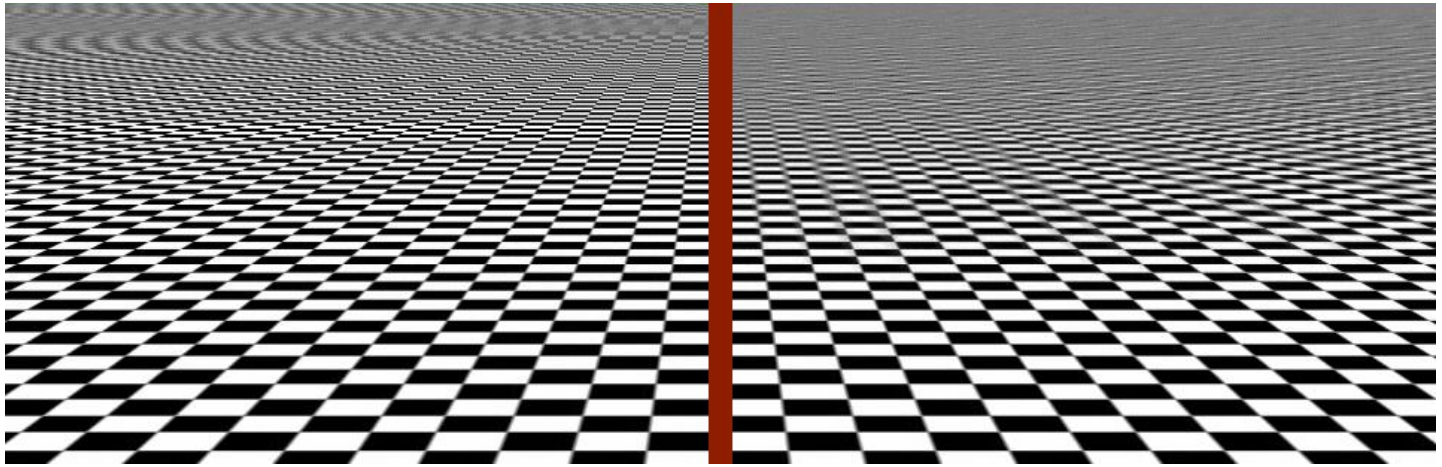


https://docs.unity.cn/cn/2021.3/Manual/texture-mipmaps-introduction.html

Prefilering

- Here is an example showing the difference when Mipmap is activated or not.
- So, why this is happening?

No Mipmap Mipmap



Prefilering

- There are two reasons for this “flickering” effect:
  - Numerical instability of floating point numbers;
  - Too high frequency signal is given.

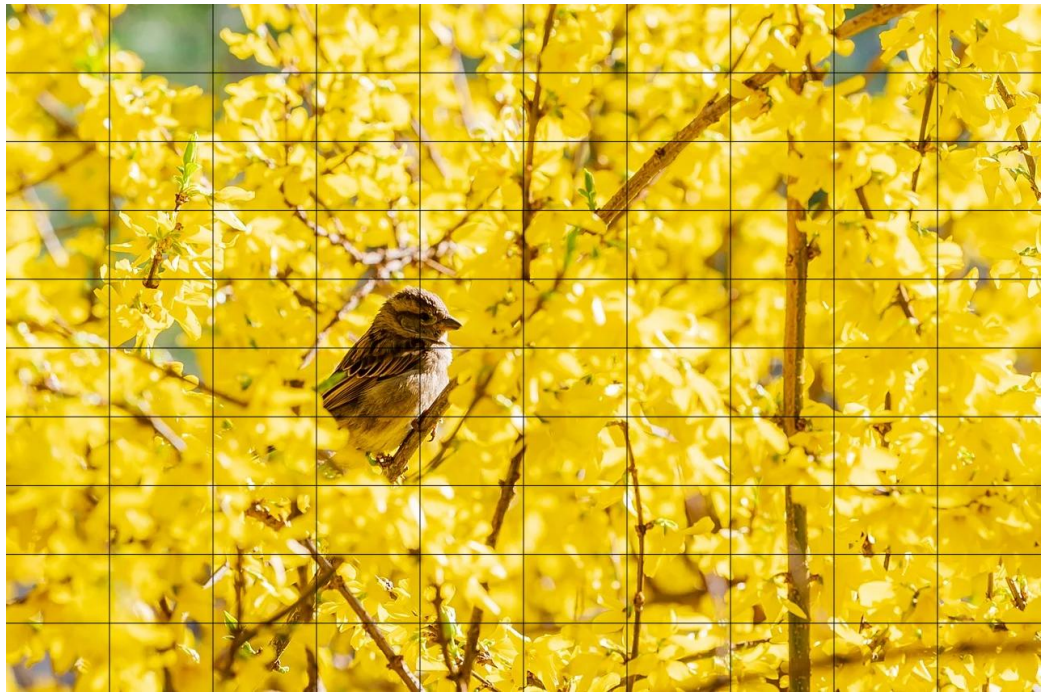
The sample grid when getting close to the object

The sample grid when getting far from the object

Prefilering

- The floating point instabity problem causes:

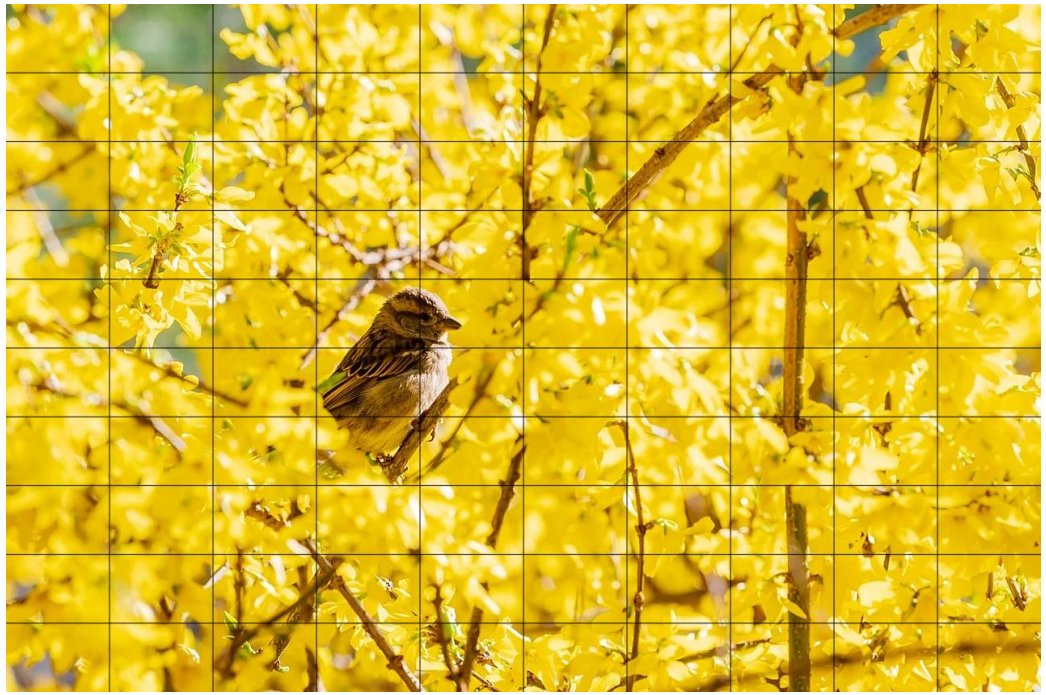
The sample grid when getting far from the object



Prefilering

- The floating point instabity problem causes:

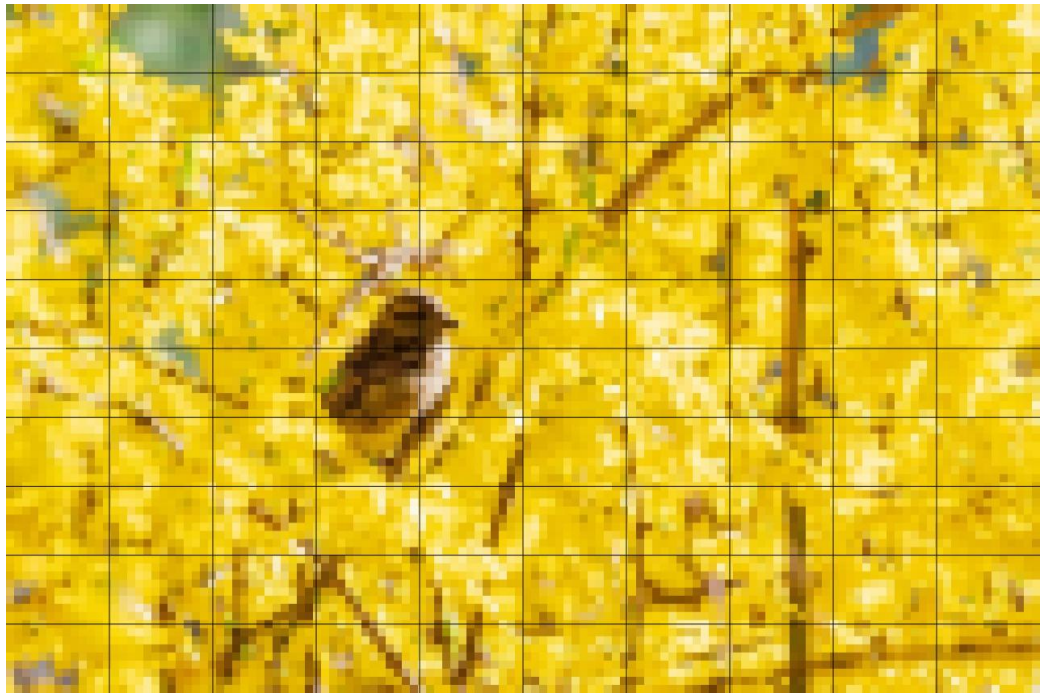
The sample grid when getting far from the object



Prefilering

- But if we have a “prefiltered” version of this image, we can switch to it when we have a low res sample grid.

The sample grid when getting far from the object



Prefilering

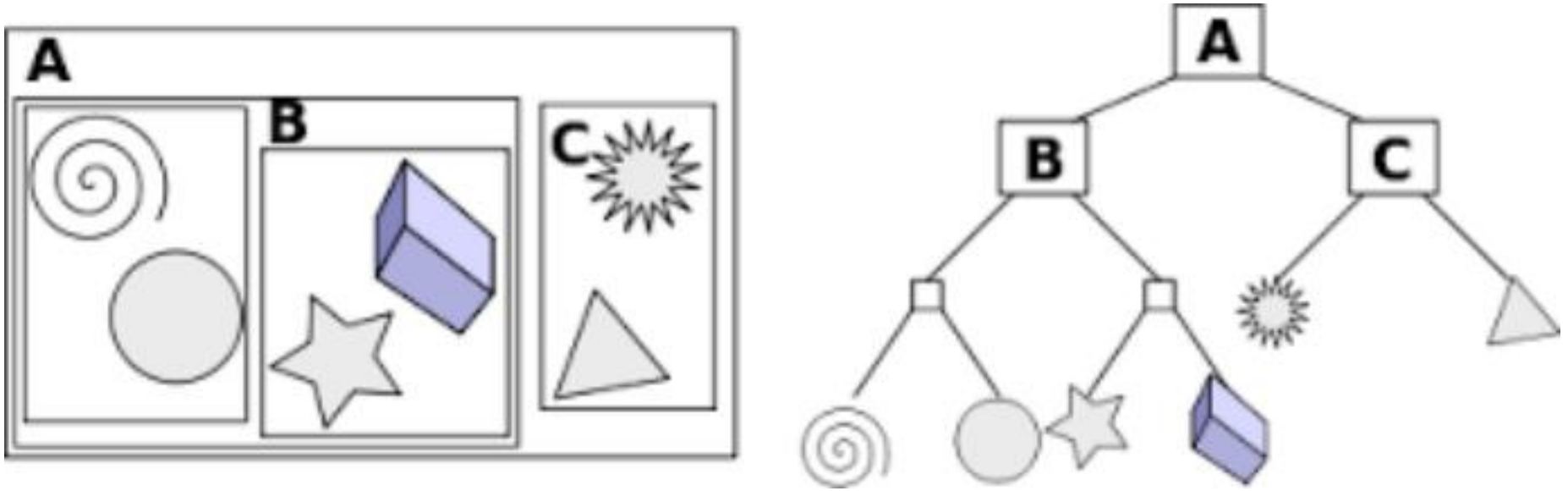
- Side to side comparison of “Mipmaped” and “Un-Mipmaped” effects in rendering tasks.

Bounding Volume Hierarchy (BVH)

- Facts about BVH:
  - It’s a data structure;
  - It’s for scene management;
  - It’s based on binary tree;
  - It’s based on Axis Aligned Bounding Box (AABB in short);
  - It’s widely used in computer graphics applications (especially ray tracing APPs)

Bounding Volume Hierarchy (BVH)

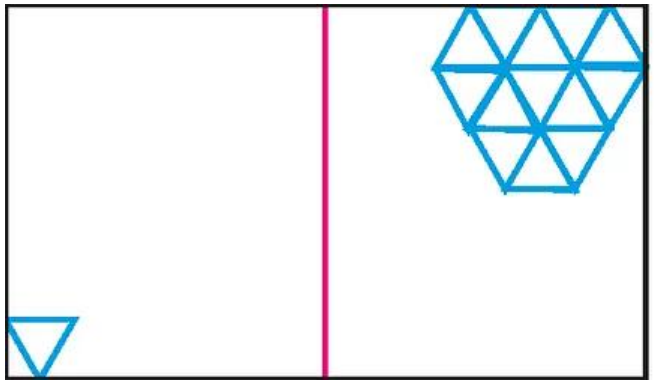
- The construction of a basic BVH tree assuming we have a 3D scene consisting of primitives (user-defined):



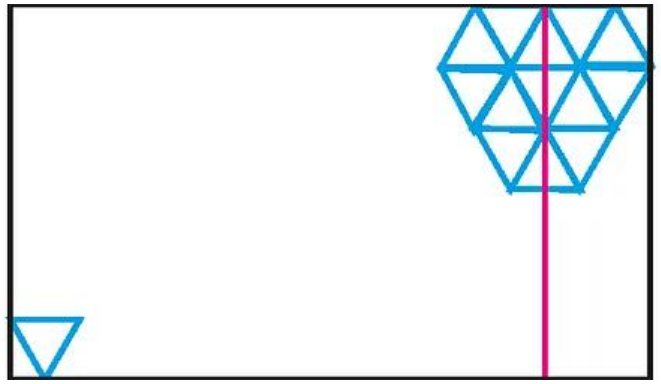
From Wikipedia

Bounding Volume Hierarchy (BVH)

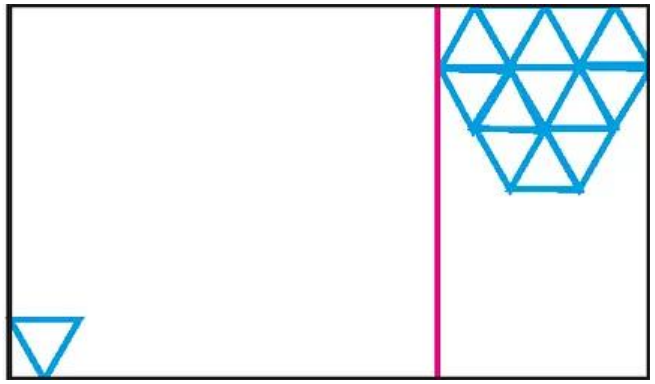
- In basic BVHs, we split the space using the middle primitive, but it’s not always the best case.



Space center as axis



Median primitive as axis

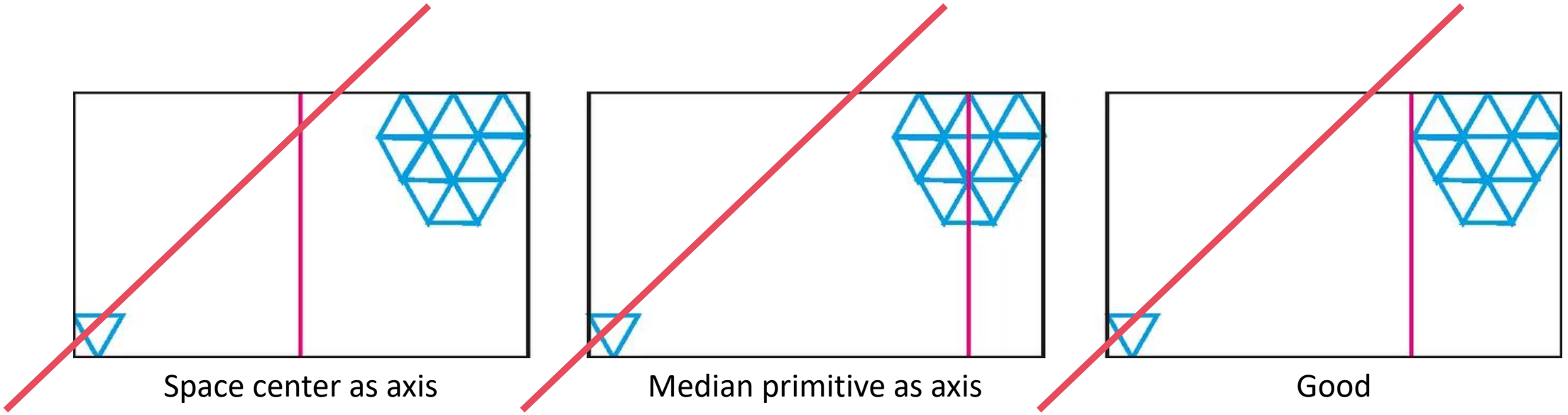


Good

https://medium.com/@bromanz/how-to-create-awesome-accelerators-the-surface-area-heuristic-e14b5dec6160

Bounding Volume Hierarchy (BVH)

- In basic BVHs, we split the space using the middle primitive, but it's not always the best case.



Space center as axis | Median primitive as axis | Good

https://medium.com/@bromanz/how-to-create-awesome-accelerators-the-surface-area-heuristic-e14b5dec6160

Bounding Volume Hierarchy (BVH)

- Researchers crave to find the best split axis when primitives are not uniformly distributed in the scene.
- For example:
  - SAH (Surface Area Heuristic for axis finding) (**used in this paper to create a BVH**)
- Outside the BVH realm, there are also many other scene management structure and tricks based on different techs:
  - KD-Tree
  - BSP (Binary Space Partitioning)
  - Octree (Quadtree in 2D cases)

SAH: https://link.springer.com/article/10.1007/BF01911006

NeRF (Neural Radiance Field, 2020)

- ***NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis***.
- It's based on volume rendering.

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: representing scenes as neural radiance fields for view synthesis. Commun. ACM 65, 1 (January 2022), 99–106. https://doi.org/10.1145/3503250

Neural Radiosity (2021)

- It take the idea from an old technique called “Radiosity”, the insight is that: Based on the rendering equation, the scene has a final state in which every surface point on every incoming and outgoing directions will follow the rendering equation.

$$L_o(x,\omega_o) = E(x,\omega_o) + \int_{\Omega} f_r(x,\omega_i,\omega_o)\cdot L_o(x',\omega_i)\cdot(\omega_i\cdot n)d\omega_i$$

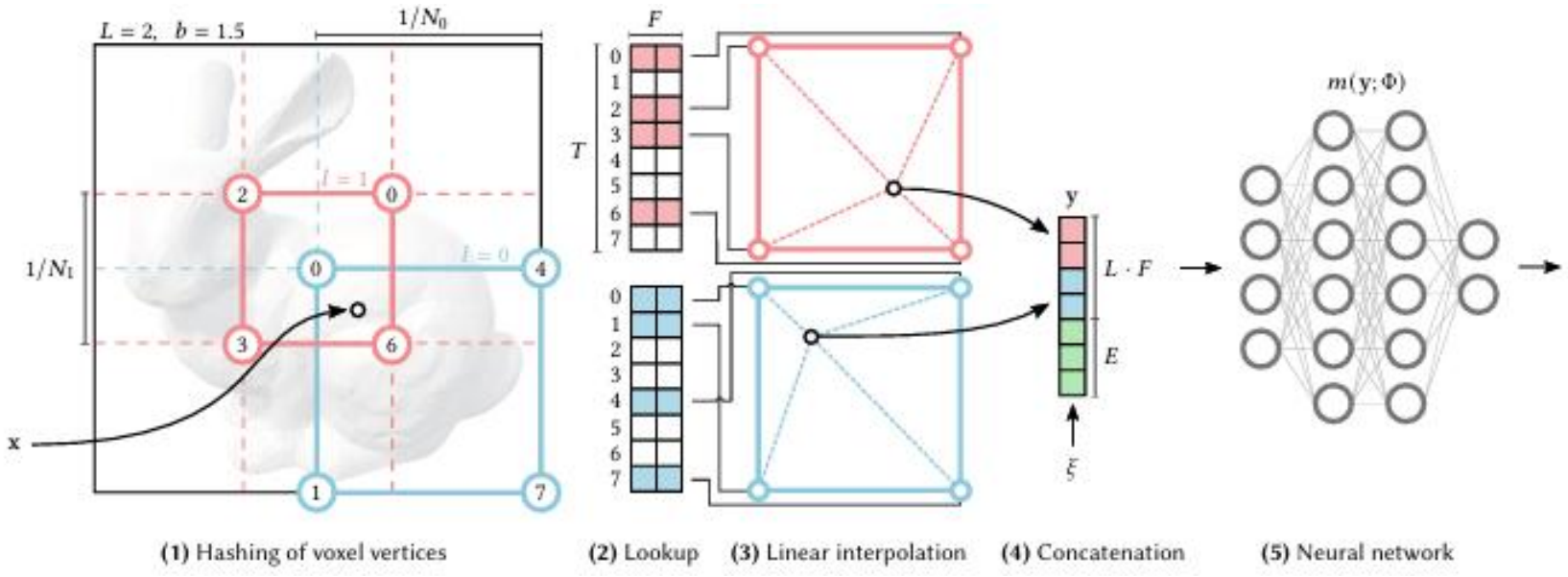
LHS RHS

LHS equals to RHS on every surface point and ω_i, ω_o!

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural radiosity. ACM Trans. Graph. 40, 6, Article 236 (December 2021), 11 pages. https://doi.org/10.1145/3478513.3480569

Instant-NGP (2022)

- It add a **learnt cache** on top of NeRF to make the training and rendering faster.
- The **learnt cache** is **multiresolution hash grid.**

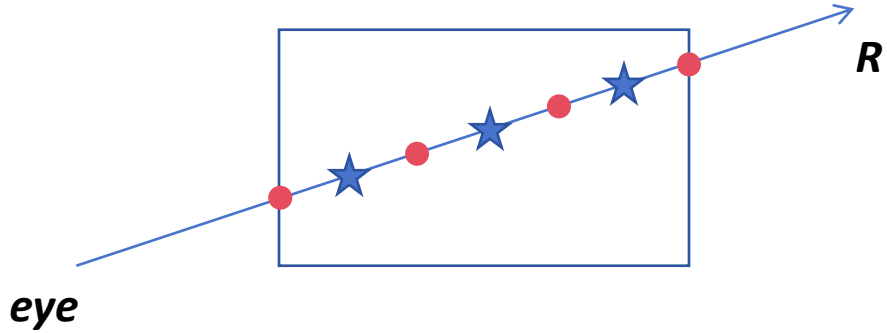


Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans. Graph. 41, 4, Article 102 (July 2022), 15 pages.

Methods

- **A learnt representation for 3D surfaces** with support for **efficient ray-intersection queries (generally, they use BVH to achieve this goal)**.
- Intuitively, we have a model ***M*** that inplicitly represent the 3D scene, then based on ***M***, the rendering task could be:
  - Generate inputs for ***M***;
  - Use inputs to evaluate/infer ***M***;
  - Render image based on infered information

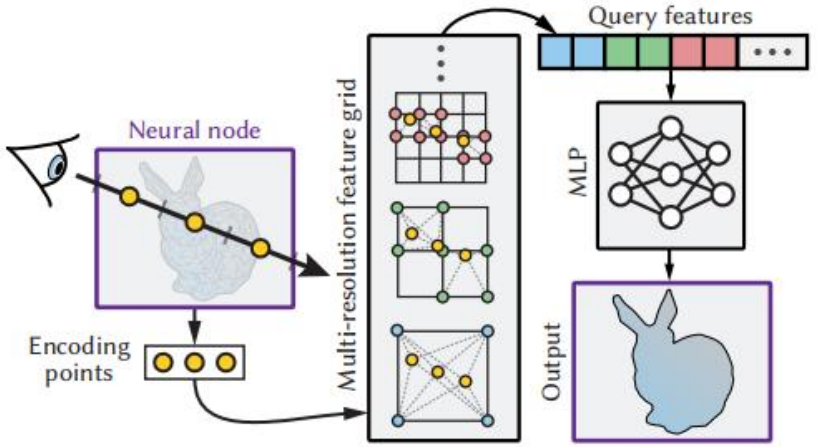
Methods



- Steps (for now):
  - Prepare a 3D scene ***S*** (triangular mesh/...);
  - Prepare a MLP;
  - Construct a conventional BVH ***Acc*** (SAH is used in the paper) based on ***S***;
  - Construct a ray ***R* = (*o*, *d*)**;
  - Do ray intersection test for ***R*** and ***Acc***;
    - Sample along the ray using stratified sampling, and get ***P* = {p_1, p_2, p_3, ...}**;
    - For each p_i in ***P***, query the underlying **multi-res hash grid** for feature vectors ***F* = {...}**;
    - Concatenate F as the input of MLP;
    - Evaluate MLP, get output (various information that is needed for rendering)

Methods

- Their pipeline. This is actually the overall pipeline of their method, which is viable. However, as they state in the paper, just doing stratified point sampling along the ray have performance issues.
- Because, if we see the actual geometry surface as the signals that the model want to learn, then the closer the sampled points is with the actual surface position, the better quality the trained model will be.



Methods

- **Refinement decision:** Let the sampled points get as close as possible to the actuall intersection points.
- One way to do this is that: make leaf nodes of the BVH smaller.

Methods

- **Refinement decision:** Let the sampled points get as close as possible to the actuall intersection points.
- One way to do this is that: make leaf nodes of the BVH smaller.
- But, when the leaf nodes become smaller, the BVH gets deeper.
- To avoid the BVH gets deeper, the authors make **another decision**: Use tree cuts to optimize the BVH, which yields **N-BVH** in this paper.

Methods

- N-BVH construction:
  - Construct a Base-BVH ***Acc*** (using SAH) based on the original geometry scene;
  - Get all the nodes along the current tree cut, ***N* = {*node*$_1$, *node*$_2$, ...}**;
  - Then calculate the node error ***Errors*** for every node in ***N***;
  - Find the indices ***Ind*** in ***Errors*** with largest Error value;
  - Replace ***node***$_{Ind}$ with its two childrens in the cut;
  - Modify the hyperparameters for Error calculation;
  - Return to step 2, until a **target node count** is reached.

Methods

- The node error is calculated via ranking heuristic:

$$r = 2\log q + \log p$$

- ***q***: Training loss on this node.
- ***p***: Probability of a random ray hitting this node. (estimated by the fraction of training rays that hit the node)
- The insight is that:
  - Priority one: nodes with large loss value need to be devided;
  - Priority two: big nodes need to be divided.

Methods

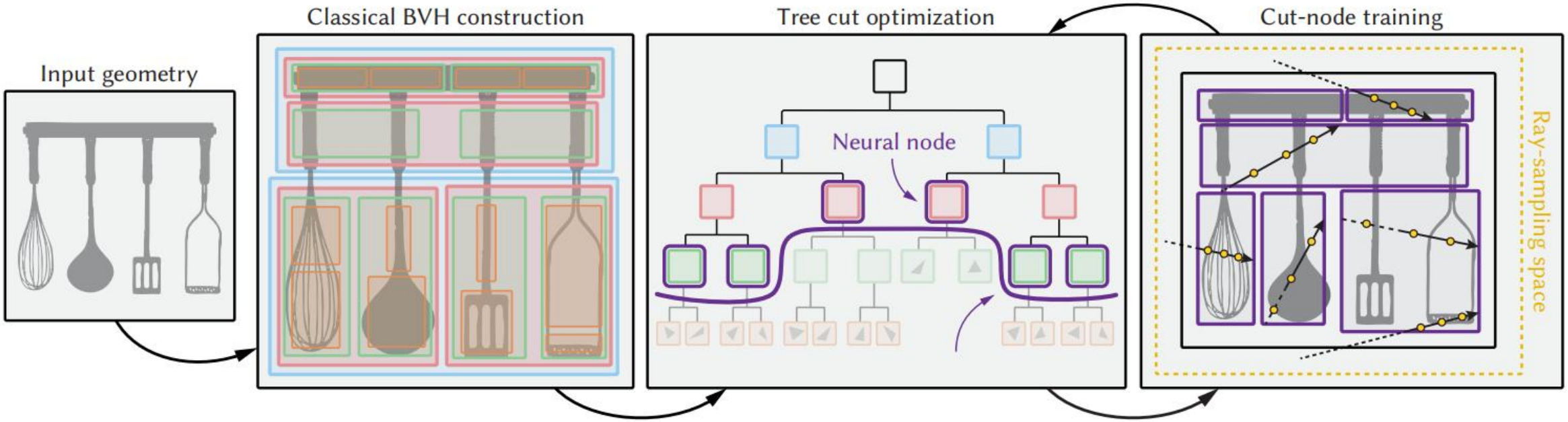
- Training process on nodes in the highest level:
  - For one sampled ray, trace the ray against nodes on cut, find the first intersected node, calculate each nodes' error, train on the first node intersected with probability ***max*(*r*/r_{max}, 0.005).**
- The output of the MLP:
  - Visibility
  - Intersection point
  - Auxiliary intersection data (normal and albedo)
- The Loss:

$$L = 2L_{vis} + 2L_{dist} + L_{norm} + L_{albedo}$$

L_{vis}: Binary cross-entropy loss
L_{dist}: L_1 loss, MAE
L_{norm}: L_1 loss, MAE
L_{albedo}: Relative L_2 loss

Methods

- The pipeline.

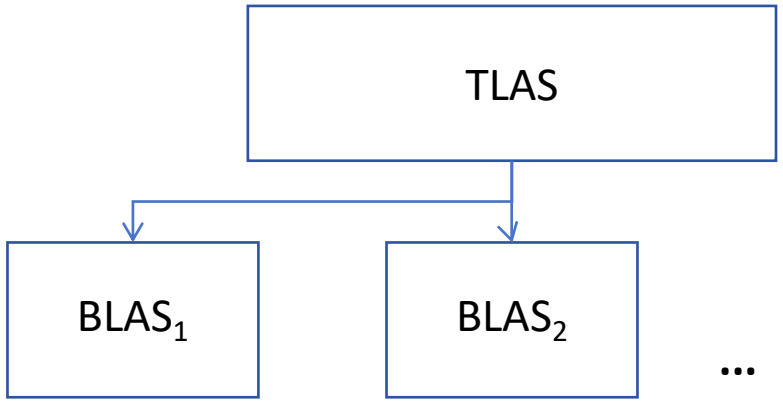


Methods

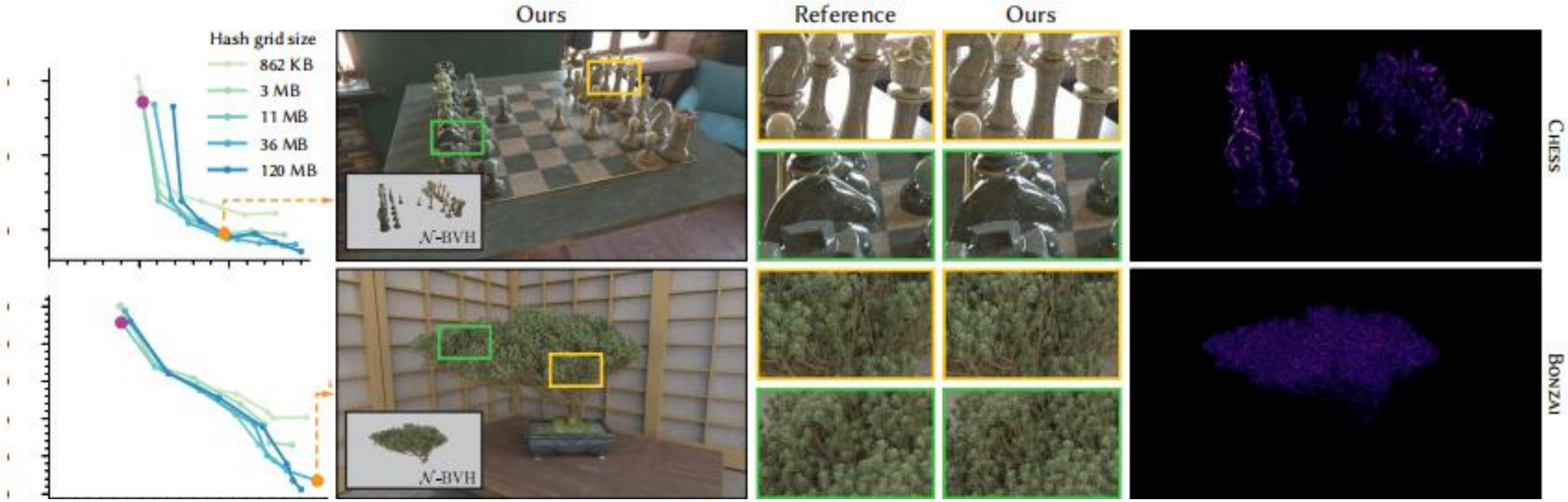
- Level of detail could be easily inplemented via N-BVH. Multiple cuts could be set and trained on an N-BVH.
- When rendering, select nodes in different cuts for LoD.

Results

- This paper implement a hybrid path tracer which has the structure like:

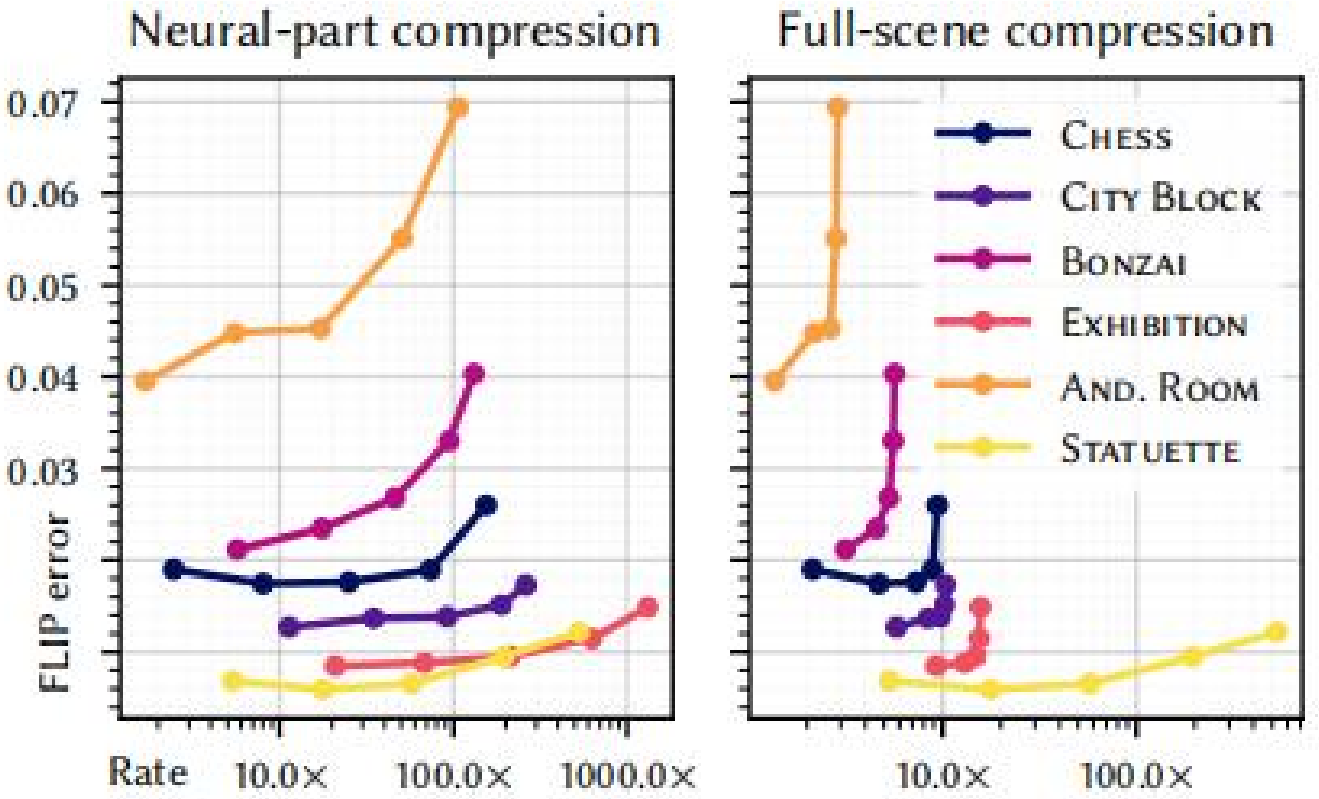


Results

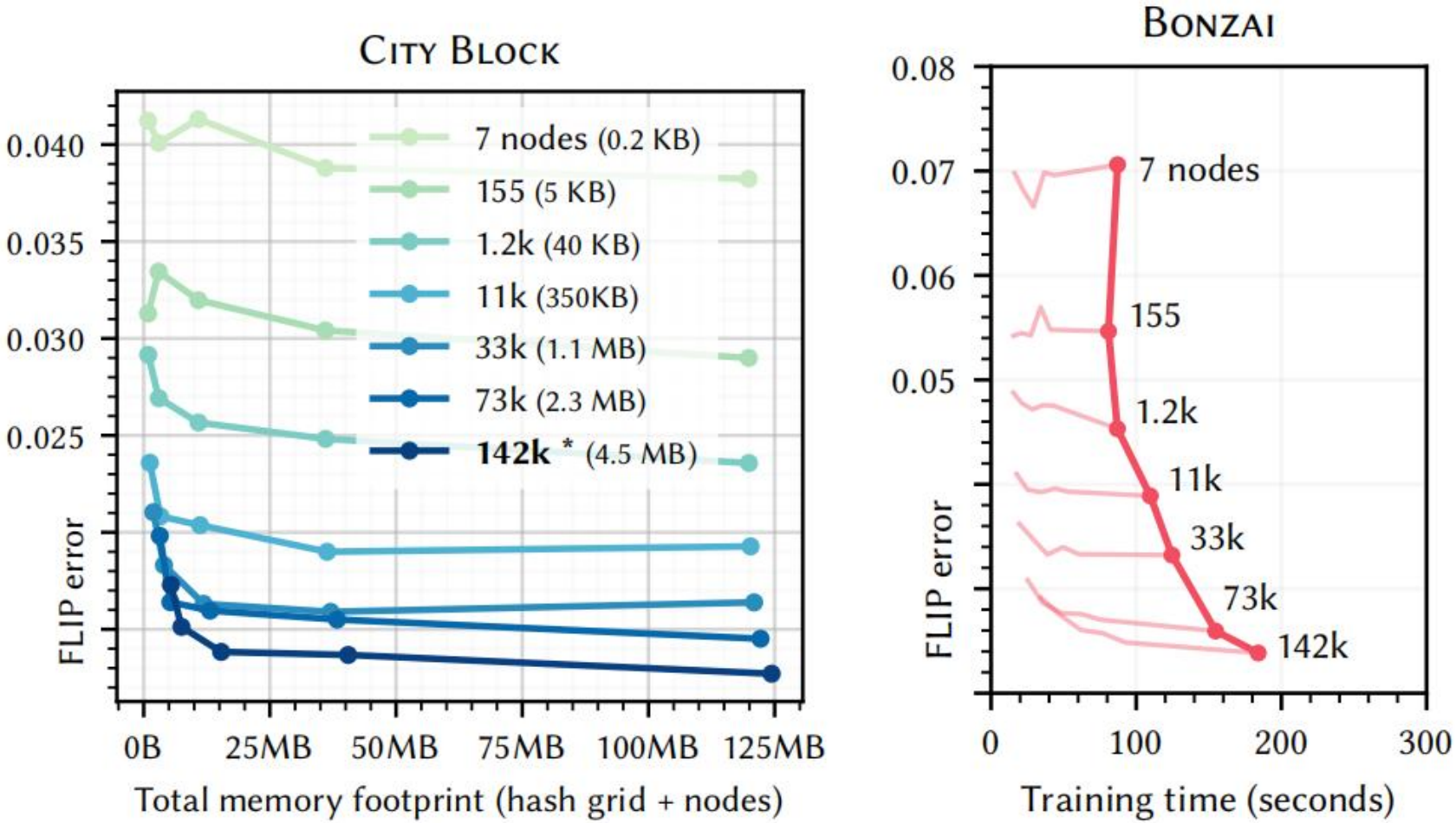


Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. Proc. ACM Comput. Graph. Interact. Tech. 3, 2, Article 15 (August 2020), 23 pages. https://doi.org/10.1145/3406183

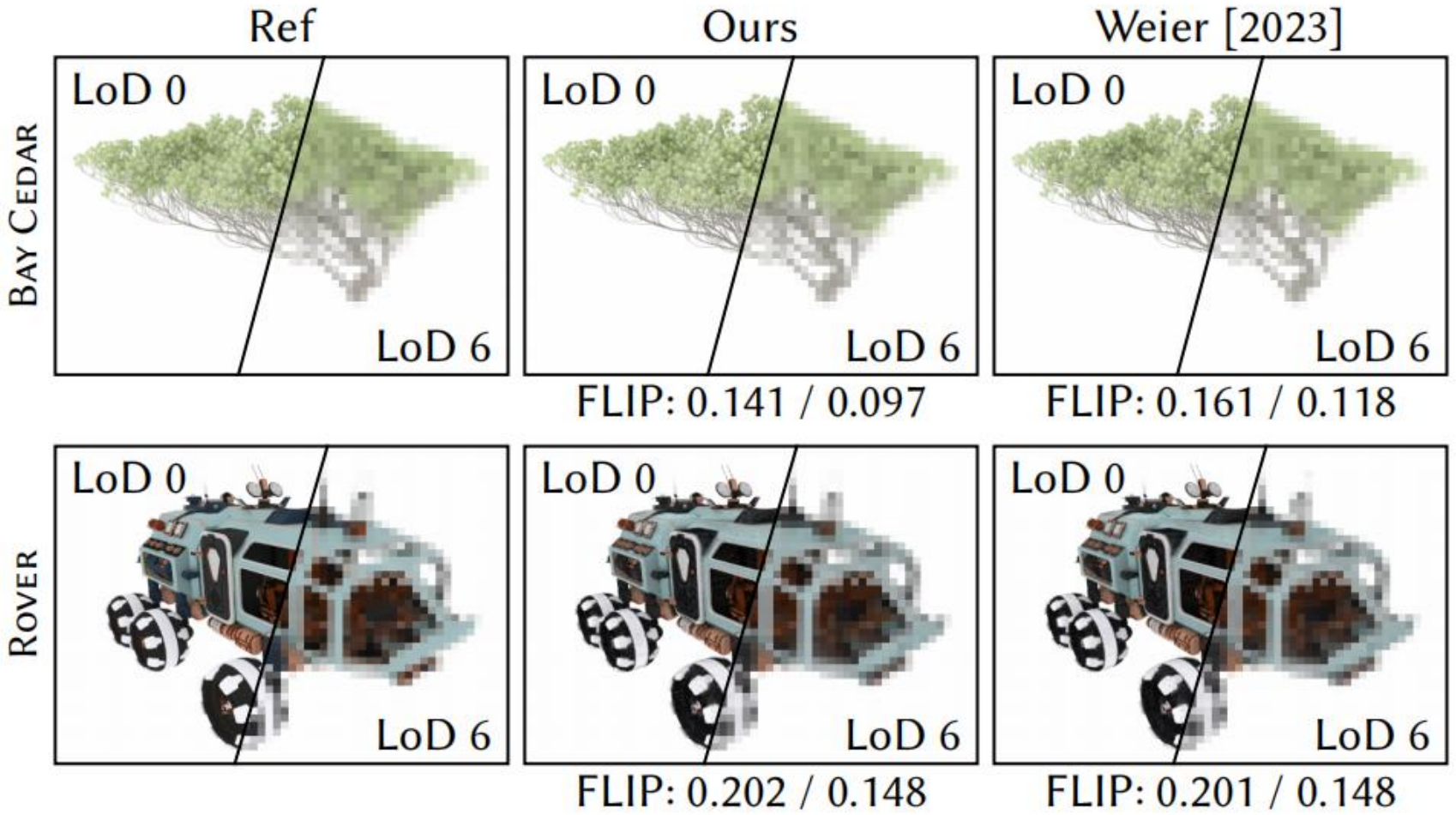
Results


Neural-part compression
Full-scene compression
0.07
0.06
0.05
0.04
0.03
FLIP error
Rate
10.0×
100.0×
1000.0×
10.0×
100.0×
Chess
City Block
Bonzai
Exhibition
And. Room
Statuette

Results

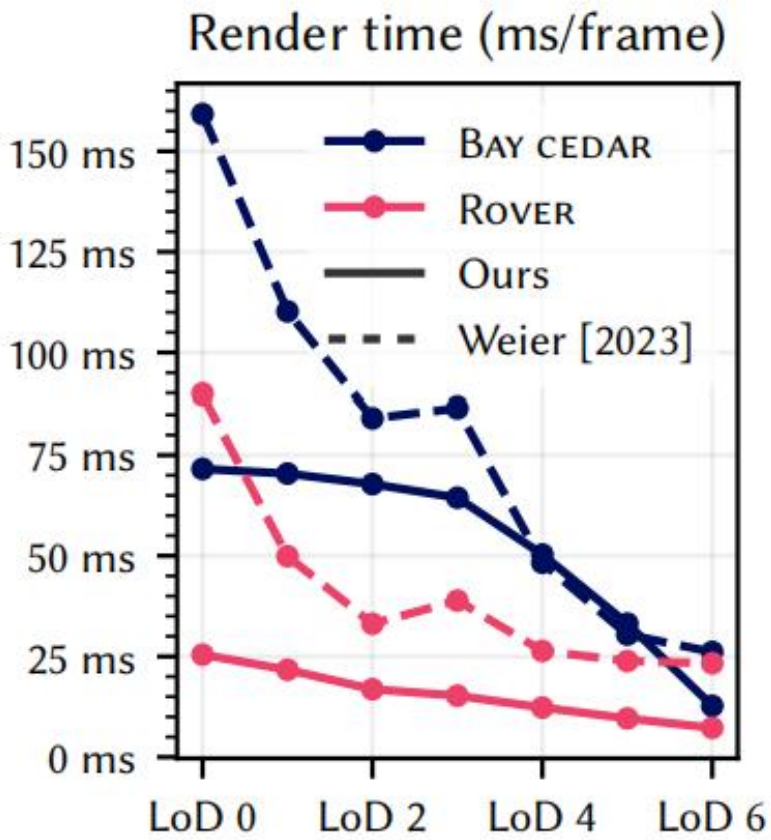

City Block
0.040
0.035
0.030
0.025
FLIP error
7 nodes (0.2 KB)
155 (5 KB)
1.2k (40 KB)
11k (350KB)
33k (1.1 MB)
73k (2.3 MB)
142k * (4.5 MB)
0B
25MB
50MB
75MB
100MB
125MB
Total memory footprint (hash grid + nodes)
Bonzai
0.08
0.07
0.06
0.05
FLIP error
7 nodes
155
1.2k
11k
33k
73k
142k
0
100
200
300
Training time (seconds)

Results

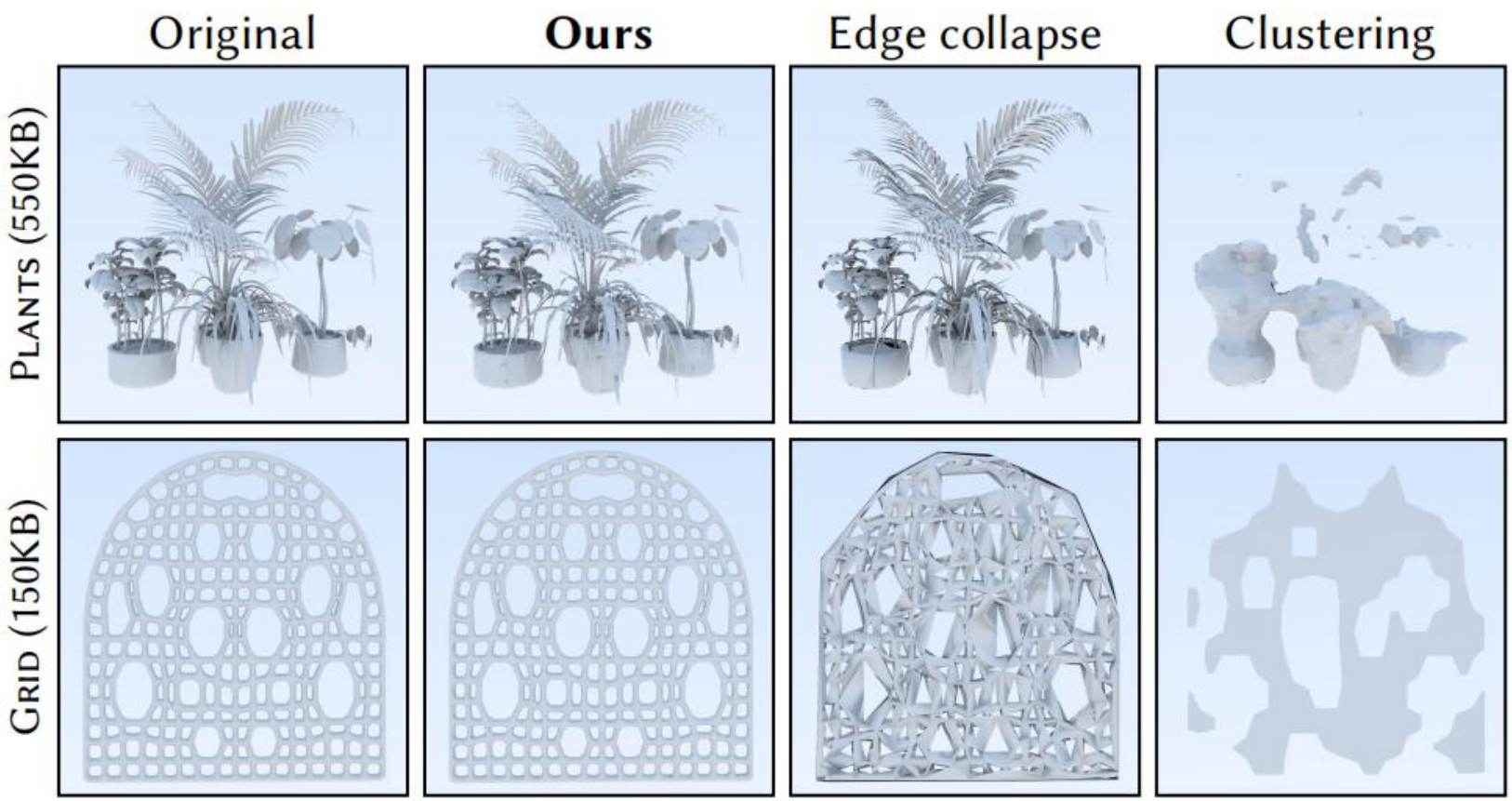

Ref
Ours
Weier [2023]
Bay Cedar
Rover
LoD 0
LoD 6
FLIP: 0.141 / 0.097
FLIP: 0.161 / 0.118
FLIP: 0.202 / 0.148
FLIP: 0.201 / 0.148

Results

Scene	Training	Network + Accel.
CEDAR		
Ours	**3:25 min**	**10.8 + 5.7 MB**
[Weier '23]	10:57 min	30.1 + 8.9 MB
ROVER		
Ours	**2:11 min**	**10.8 + 0.7 MB**
[Weier '23]	6:14 min	15.1 + 9.5 MB



Results


Original
Ours
Edge collapse
Clustering
PLANTS (550KB)
GRID (150KB)