Deep Hybrid
Order-Independent Transparency

Grigoris Tsopouridis[1], Ioannis Fudos[1], Andreas-Alexandros Vasilakis[1,2]

[1] Department of Computer Science and Engineering, University of Ioannina, Greece
[2] Department of Informatics, Athens University of Economics and Business, Greece

Co-financed by Greece and the European Union

The Project is co-funded by the European Union
and by National Funds of Greece & Albania

Transparency Effects

Glass **transmission** and **refraction**

Foliage **partial coverage**

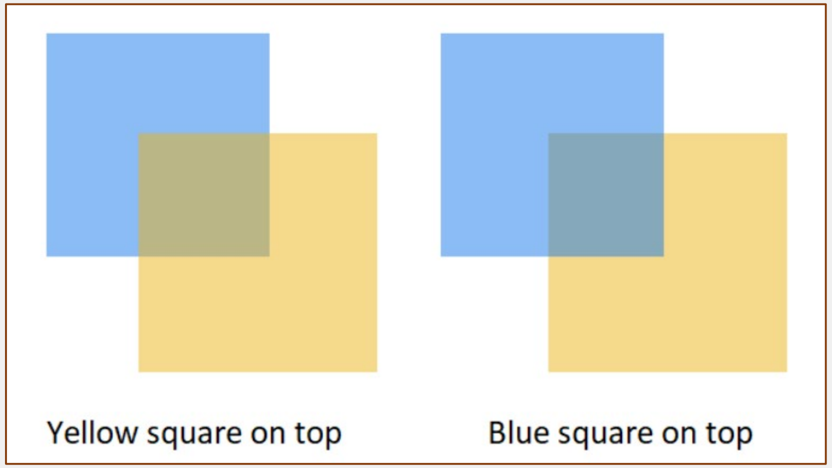
Volumetric lighting

Handling Transparency in Rasterization

- **Problem**: Rasterization generates more than one out-of-order fragments per pixel
  - Z-buffer, single-fragment visibility determination
  - Alpha blending is not **correct**

- **Correct transparency composition** requires fragments generated in depth order [PD84]

Compositing order matters!



$$v_i = \begin{cases} \alpha_1 & \text{if } i = 1 \\ \alpha_i \times \left(1 - \sum_{j=1}^{i-1} v_j\right) & \text{if } i > 1 \end{cases}$$

$$Color = \sum_{i=1}^{k} (C_i \times v_i)$$

[PD84]

[PD84] Porter & Duff: Compositing digital images, SIGGRAPH ’84.

Order-Independent Transparency (OIT)

- **OIT** resolves transparency without explicit fragment ordering
- **Classification**
  - **Exact**: Buffer-based methods (multifragment rendering)
  - **Approximate**: Faster, inaccurate methods
    - e.g.: Alter the compositing operator

Common OIT Solution Strategy

Multifragment rendering [VVP20]

- A-buffer
  - Stores all scene information
- k-buffer
  - Stores a fraction of scene information

[VVP20] Vasilakis, et al.: A survey of multifragment rendering, EG STAR 2020.

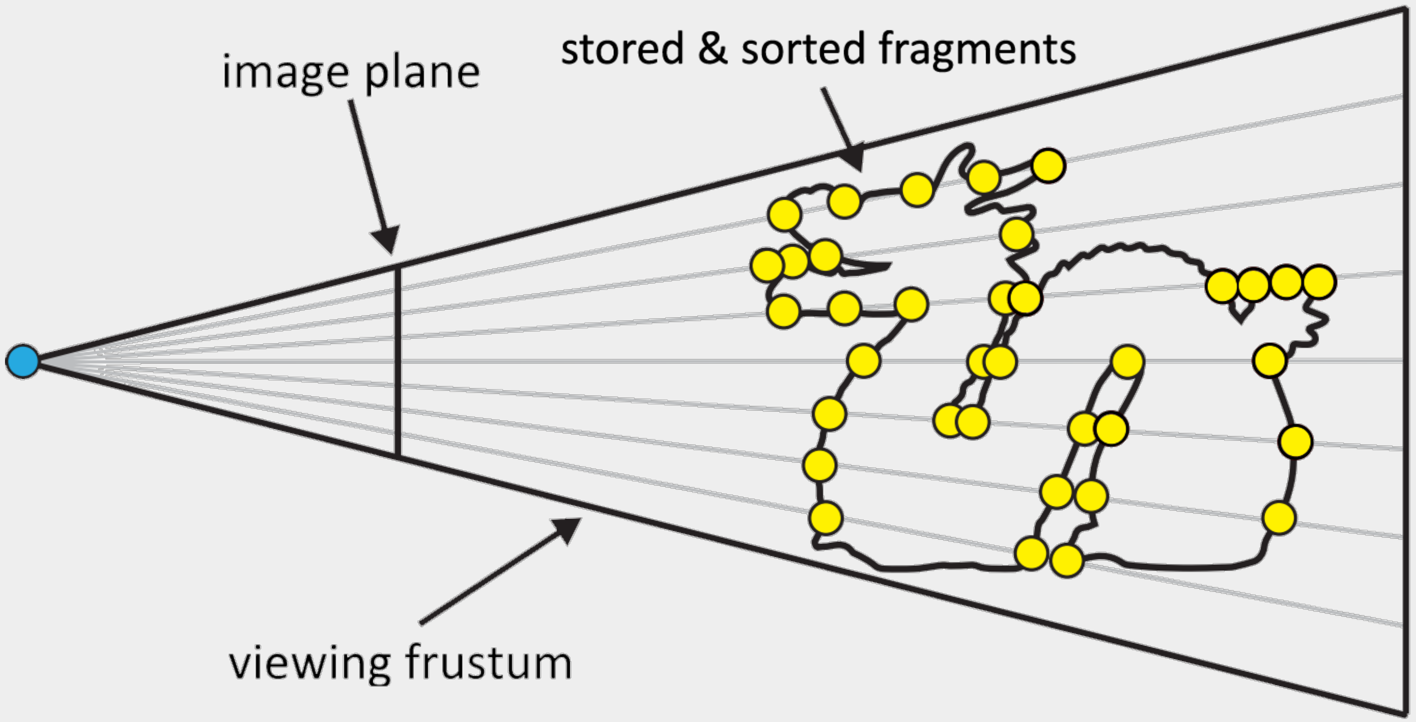
A-buffer: Correct OIT result

Multifragment rendering [VVP20]

- A-buffer
  - Stores all scene information
- k-buffer
  - Stores a fraction of scene information

[VVP20] Vasilakis, et al.: A survey of multifragment rendering, EG STAR 2020.

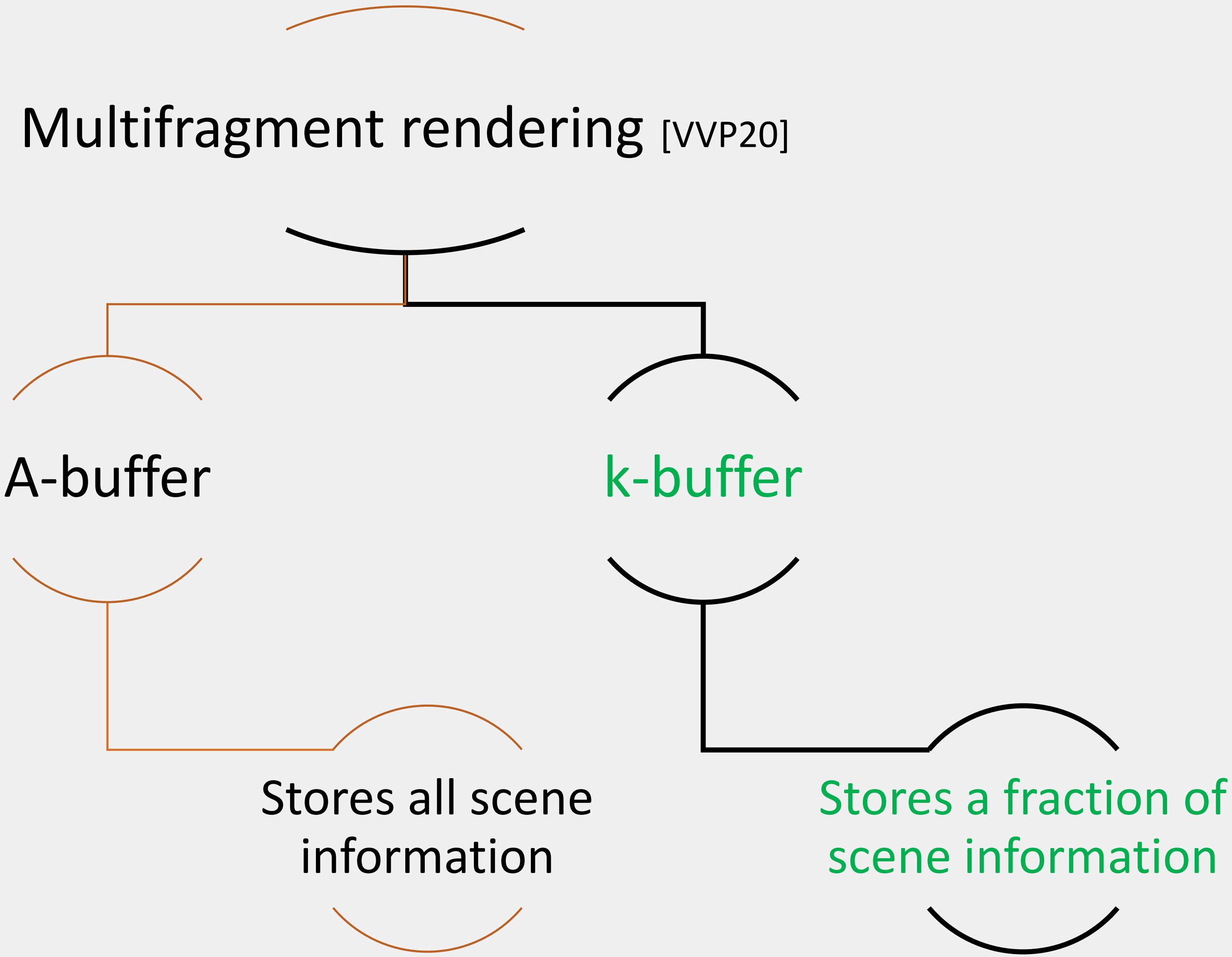
A-buffer: Store all fragments then sort them [Car84]



- **Perfect quality**
  - Operate on all sorted fragments
- **Memory overflow**
  - Unbounded storage allocation
- **Slow sorting**
  - Cache overflow and latency issues

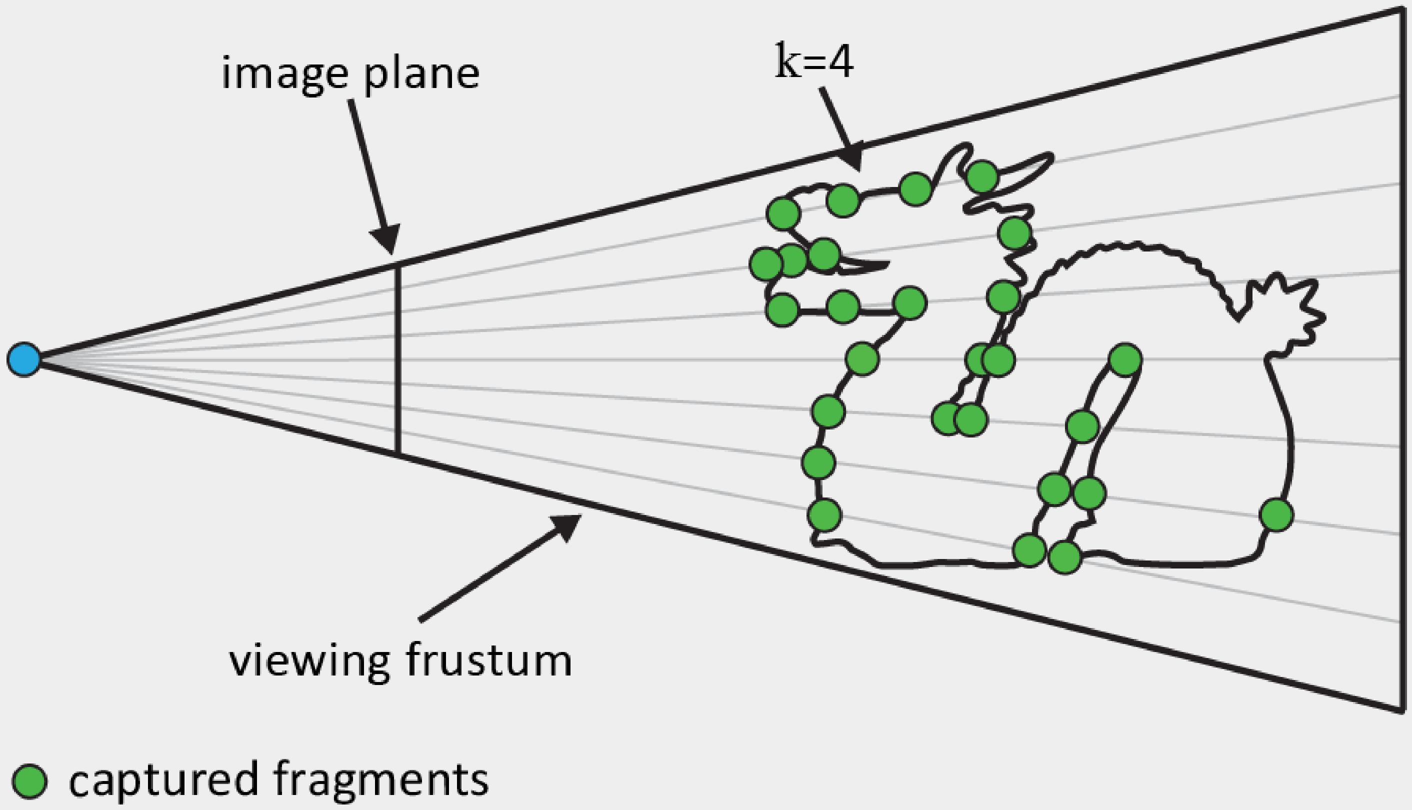
[Car84] Carpenter: The A-buffer, an antialiased hidden surface method., SIGGRAPH 1984.

k-buffer: Partly exact OIT result



[VVP20] Vasilakis, et al.: A survey of multifragment rendering, EG STAR 2020.

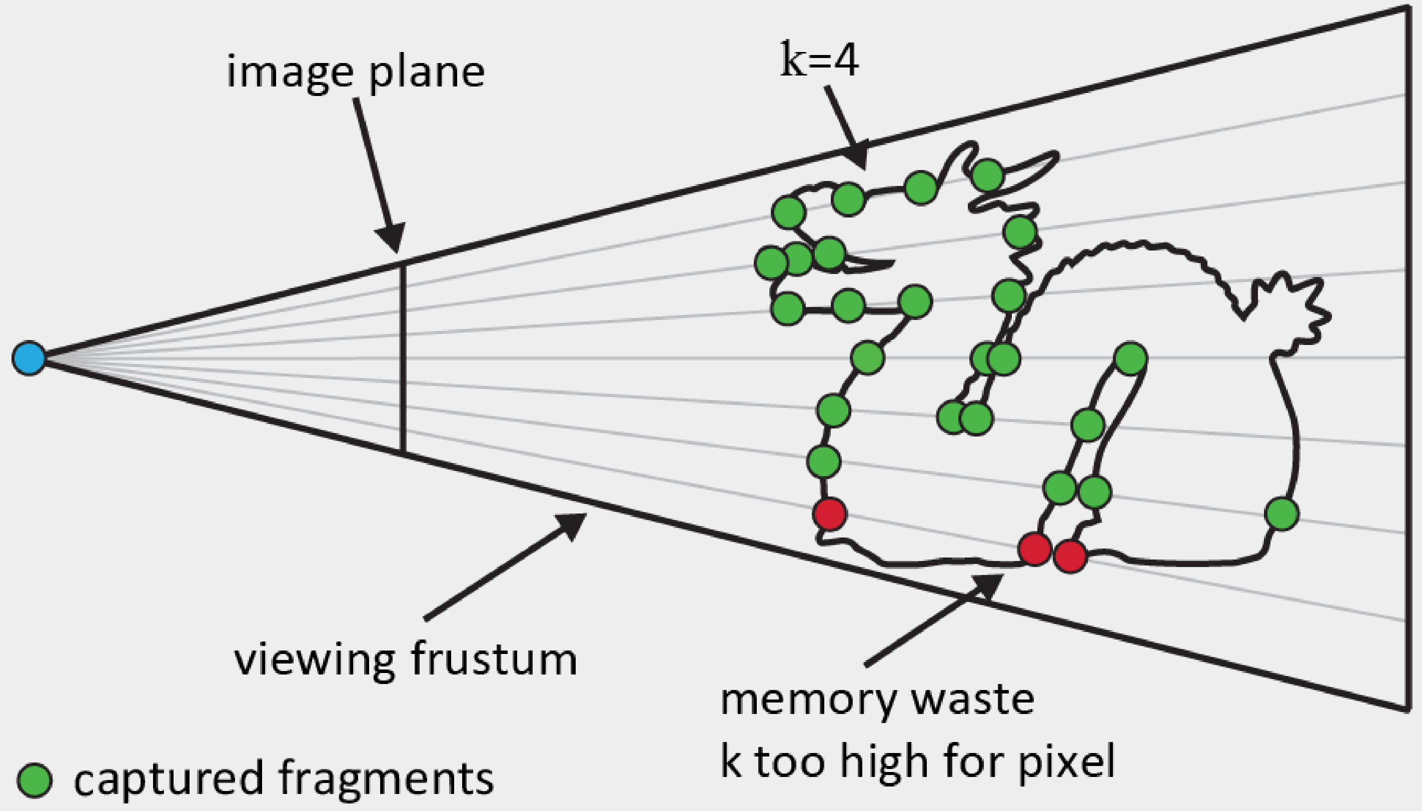
k-buffer: Store and sort *k* fragments [BCL*07]



- Captures a **subset** of all fragments
  - Usually, the k-closest to camera
- Requires a **fixed, pre-defined memory**
  - **Fixed** (global value)
  - **Variable** (per-pixel value)

[BCL*07] Bavoil, et al.: Multi-fragment effects on the GPU using the k-buffer, I3D 2007.

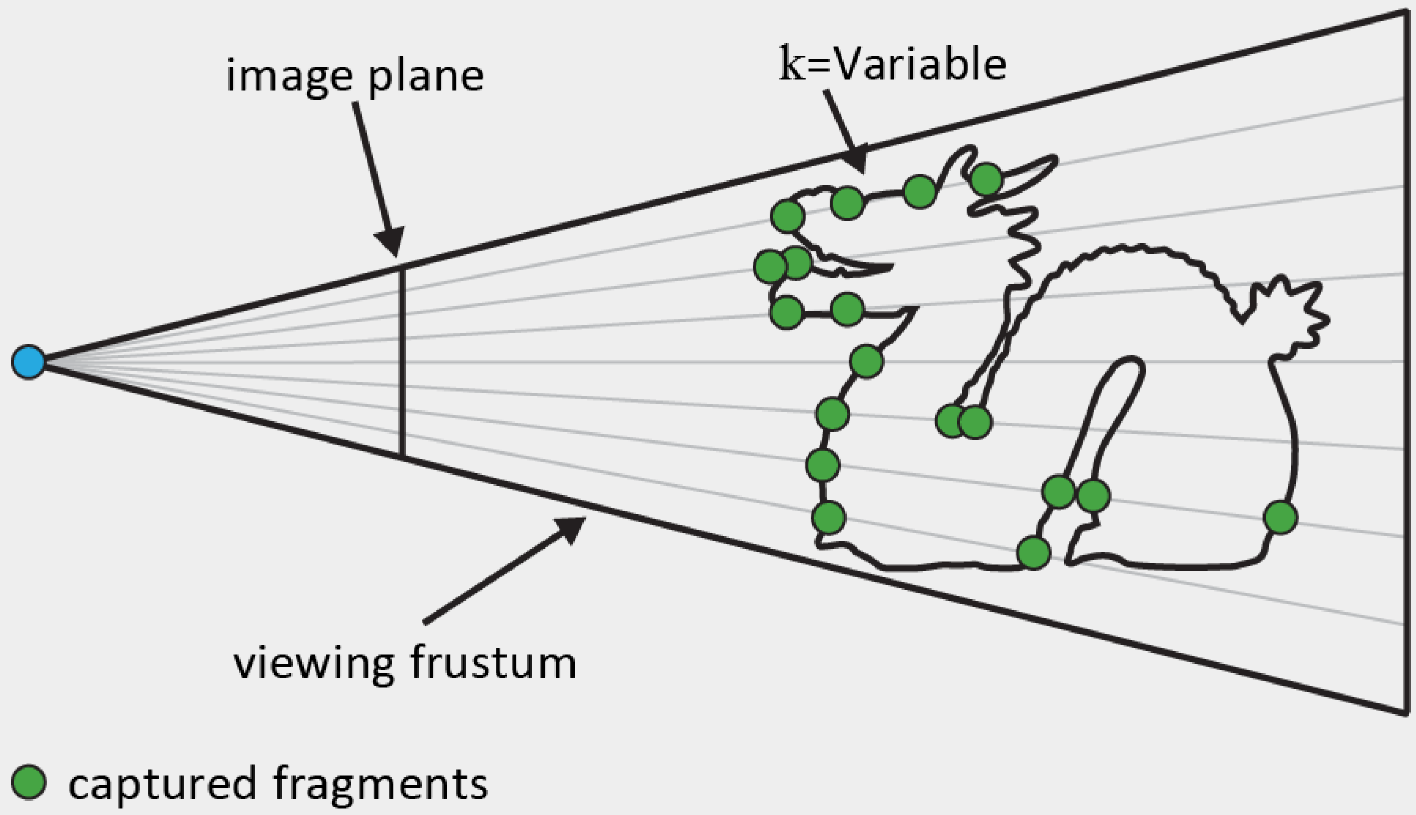
Challenge to find a good global k value



- **Fine-tuning global k-value**
  - Set **low**: view-dependent artifacts
  - Set **high**: unused memory
- **Automatic solution** based on depth complexity histogram analysis [VPF15]
  - Alleviate some problems, not all!

[VPF15] Vasilakis, et al.: k^{+}-buffer: An efficient, memory-friendly and dynamic k-buffer framework, IEEE TVCG 2015.

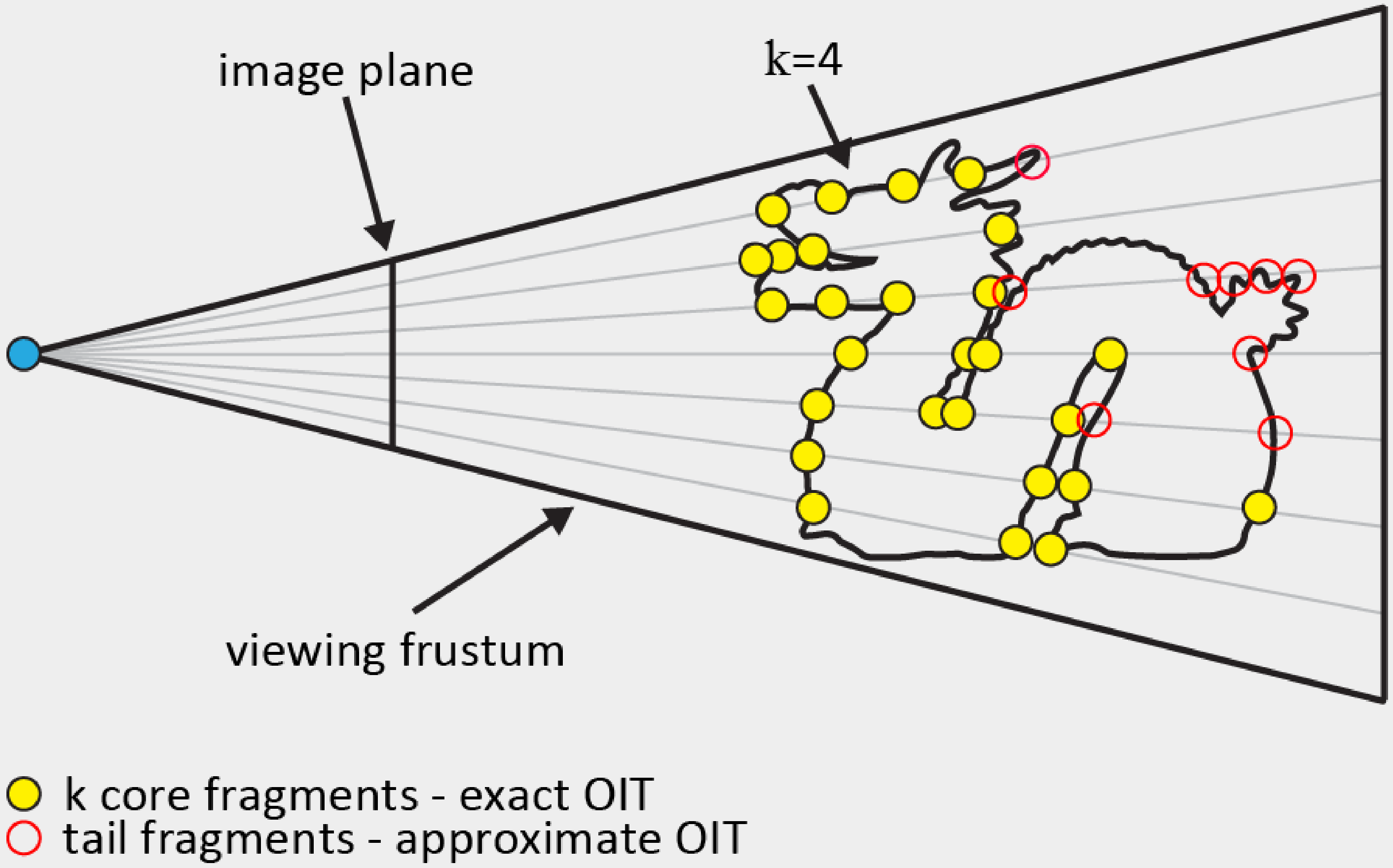
Variable k-buffer [VVPM17]



- **Better image quality** for same allocated memory bandwidth
  - Uses more memory for the "**more important**" pixel areas of the image
  - Exact memory allocation in a **global continuous buffer**
- **Performance loss**
  - Additional rendering pass
  - Thread divergence, higher fragment complexities

[VVPM17] Vasilakis, et al.: Variable k-buffer using importance maps, EG Short Papers 2017.

Hybrid Transparency [MCTB13]



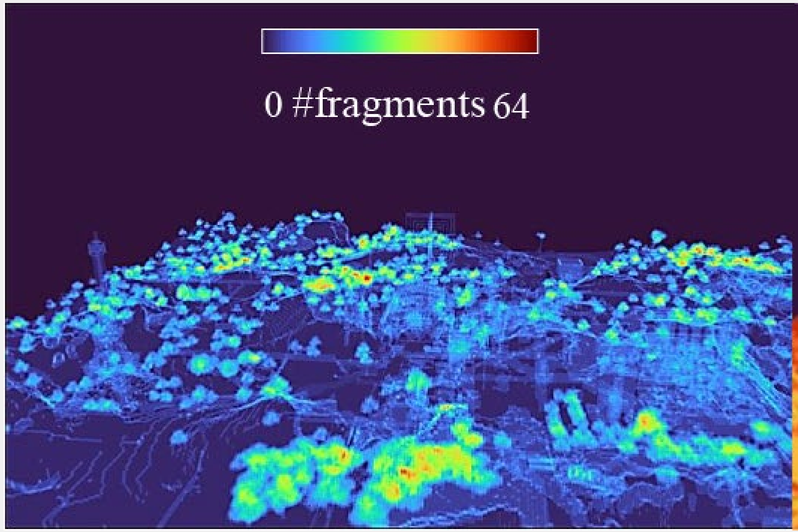
- **Core**
  - Uses a traditional k-buffer
  - Exact, but slow, OIT of near fragments
- **Tail**
  - Rest fragments are blended
  - Fast, but approximate, OIT of far fragments

- OIT is the **combination** of the above

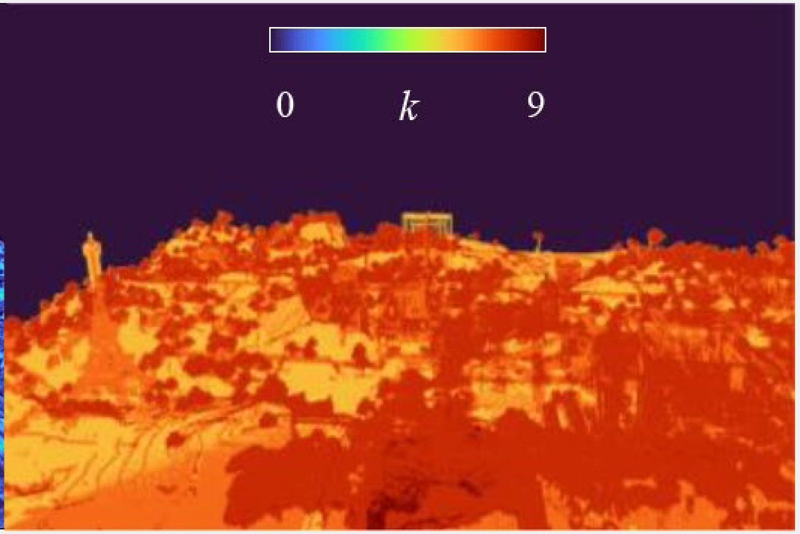
[MCTB13] Maule, et al.: Hybrid Transparency, I3D 2013.

Can we predict better per-pixel k values?

- **Deep Hybrid Order-Independent Transparency**
  - Predict pixel importance with a **deep learning** prediction mechanism, under a fixed and pre-defined memory budget
  - Exploit **Hybrid Transparency** strategy to further improve visual quality



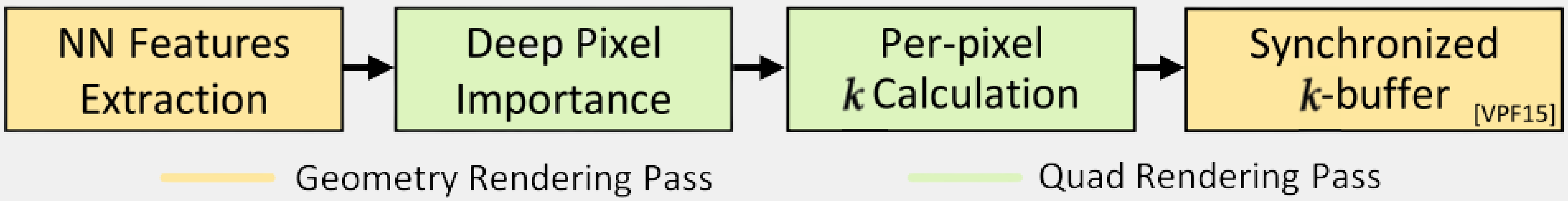
Fragment distribution heatmap



Generated per-pixel k-values

Deep k-buffer rendering pipeline

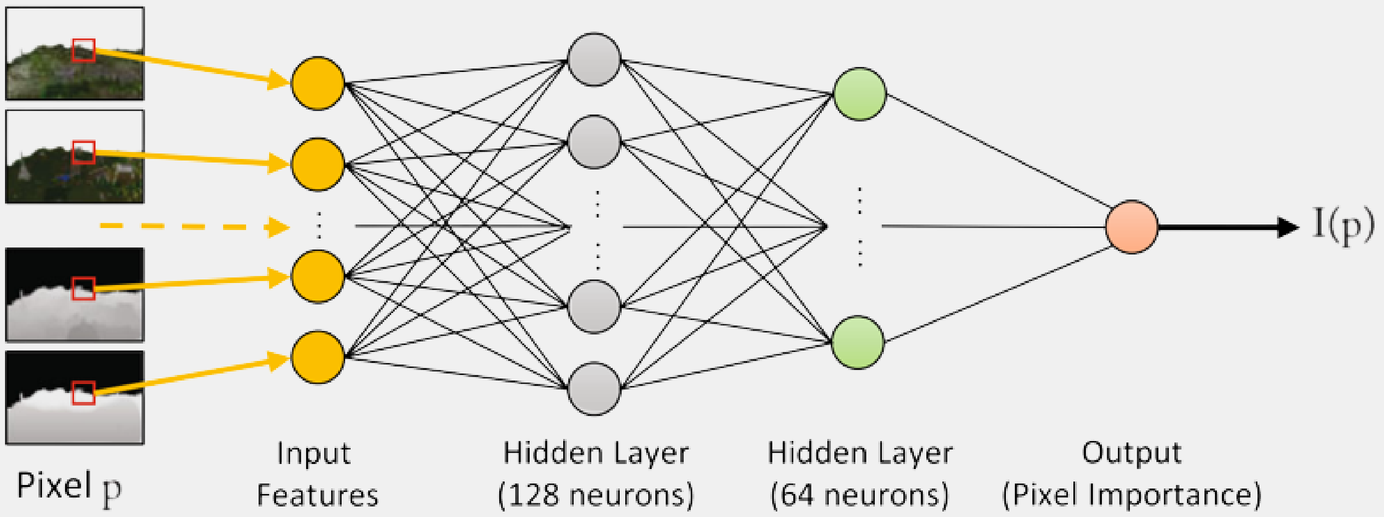
- A fast geometry pass is used to **extract NN inputs** from the scene
- Per-pixel **importance** is then predicted from the NN which is then
  - Used to compute a variable pixel k-value
  - Memory (k values) allocated in areas considered more important
- **Synchronized k-buffer** is used to store and sort the **core** fragments [VPF15]
- **Tail** fragments are accumulated with a quick **Weighted Average** approximation [BM08]



[VPF15] Vasilakis, et al.: k⁺-buffer: An efficient, memory-friendly and dynamic k-buffer framework, IEEE TVCG 2015.
[BM08] Bavoil & Mayers: Order independent transparency with dual depth peeling, Nvidia report, 2008.

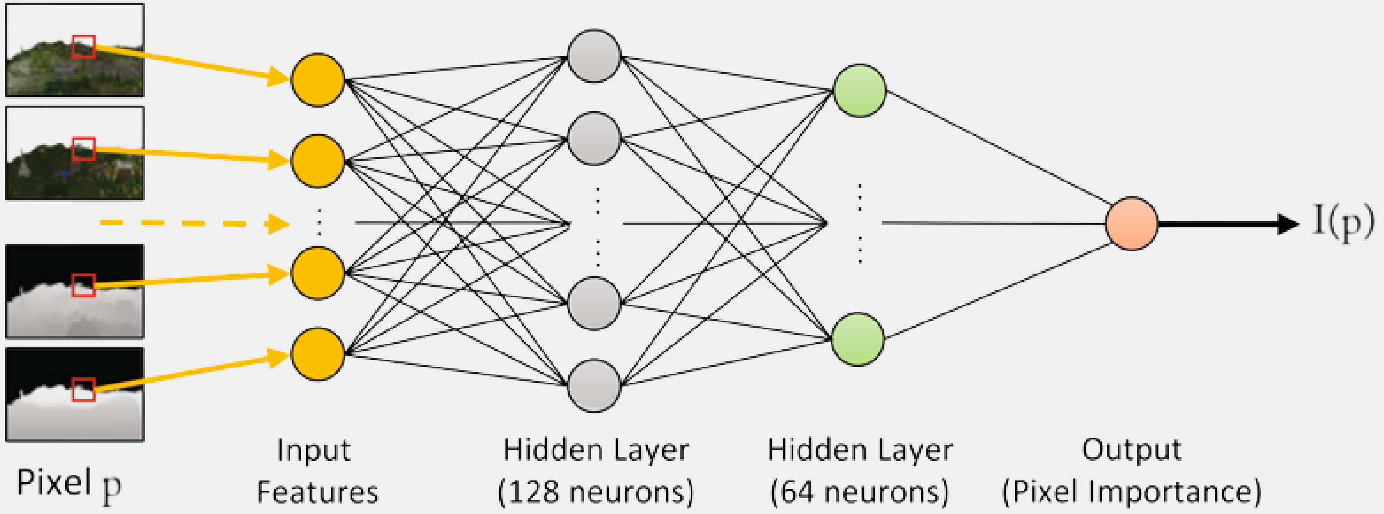
Neural Network Architecture

- Simple neural network with two hidden layers of 128 and 64 neurons
- ReLU hidden layer activation function
- Sigmoid activation function used for output layer
  - Expresses pixel importance *I(p)* ∈ [0,1] values
- SGD optimizer
- 12 float input features



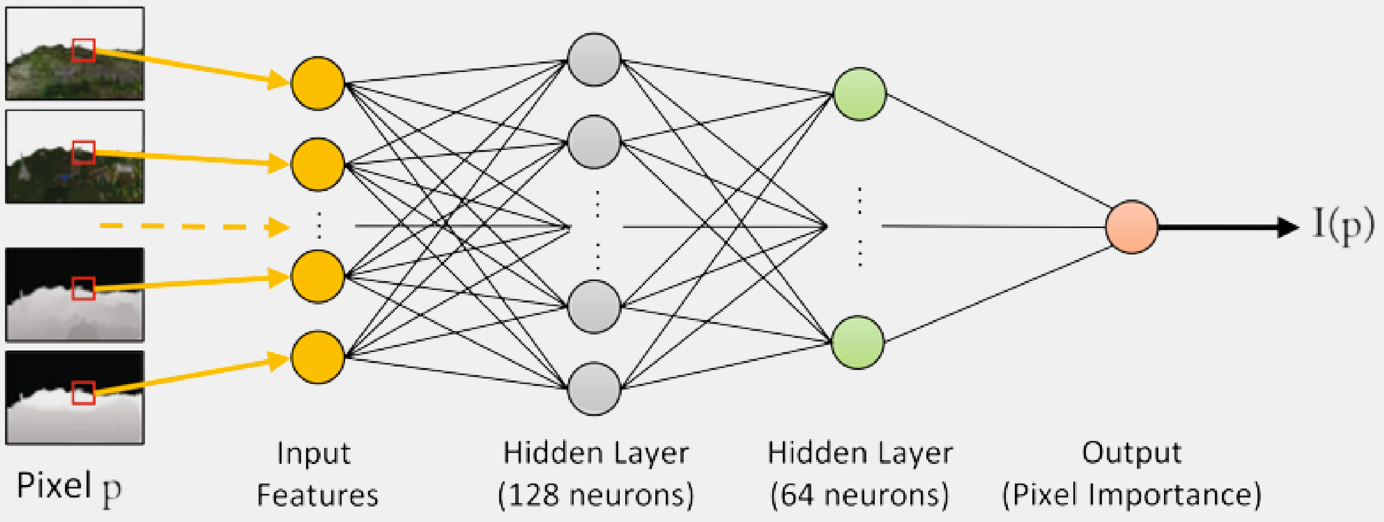
Neural Network Input Features

- **Image-based**:
  - Allocated fragments (memory budget M) divided by total number of fragments of the scene
  - Nearest and farthest scene fragment depths
- **Per-pixel**:
  - Nearest and farthest pixel fragment depths
  - Diffuse color (RGB) of the nearest fragment
  - Average diffuse color (RGB) of pixel fragments
  - Number of pixel fragments divided by the total number of fragments of the scene



Neural Network Training

- Trained from 8 fully transparent scenes with varying depth complexity and multiple scene views
- Desired outputs (**optimal pixel importance**) produced offline using a greedy algorithm that computes the optimal fragment distribution



But how we find the “optimal” pixel importance?

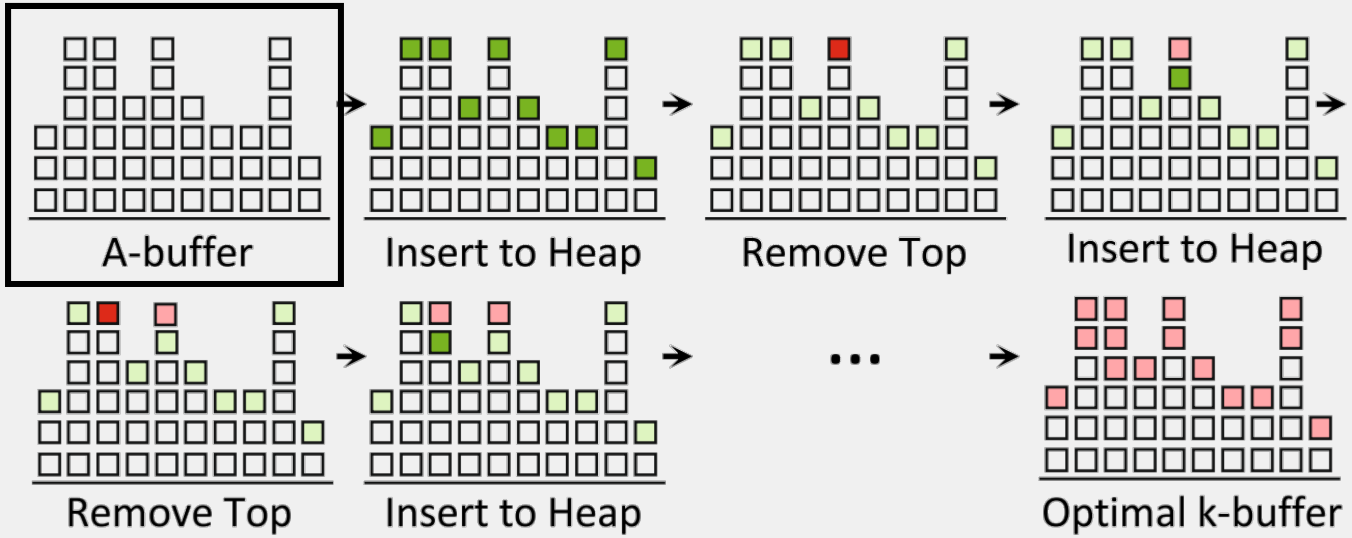
- **Optimal fragment distribution**

Distribute fragment space (based on the desired memory allocation) to each pixel in order to minimize the MSE of Hybrid Transparency compared to exact A-buffer method

- **Optimal per-pixel importance**

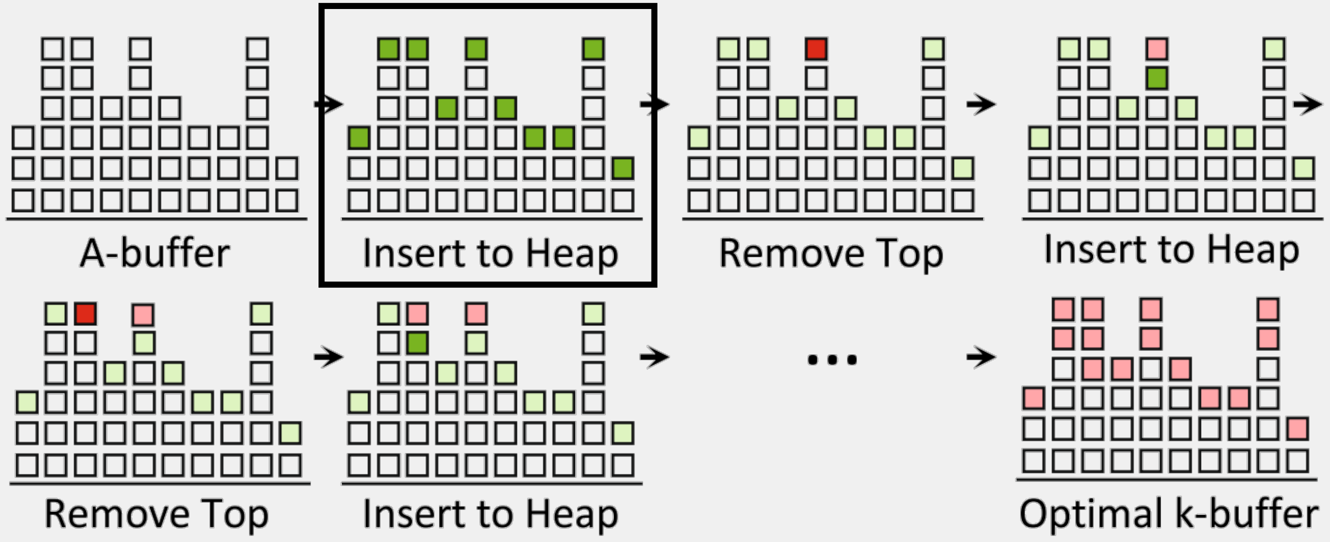
Divide the optimal fragment distribution by the desired total fragments (memory limit)

Optimal k-buffer method



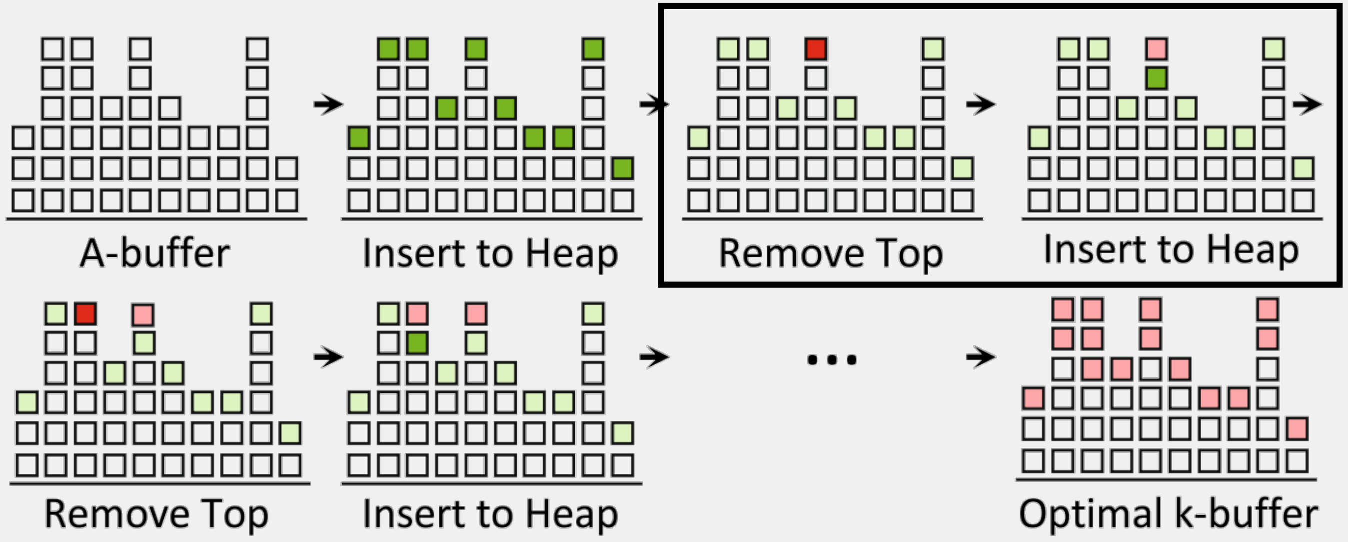
1. Starting from **A-Buffer, set** the optimal fragment distribution equal to the A-buffer distributions and a desired **memory** size, and **remove** at each iteration the **farthest** from the viewer **fragment** that causes the **minimum perceptual error** increase.

Optimal k-buffer method



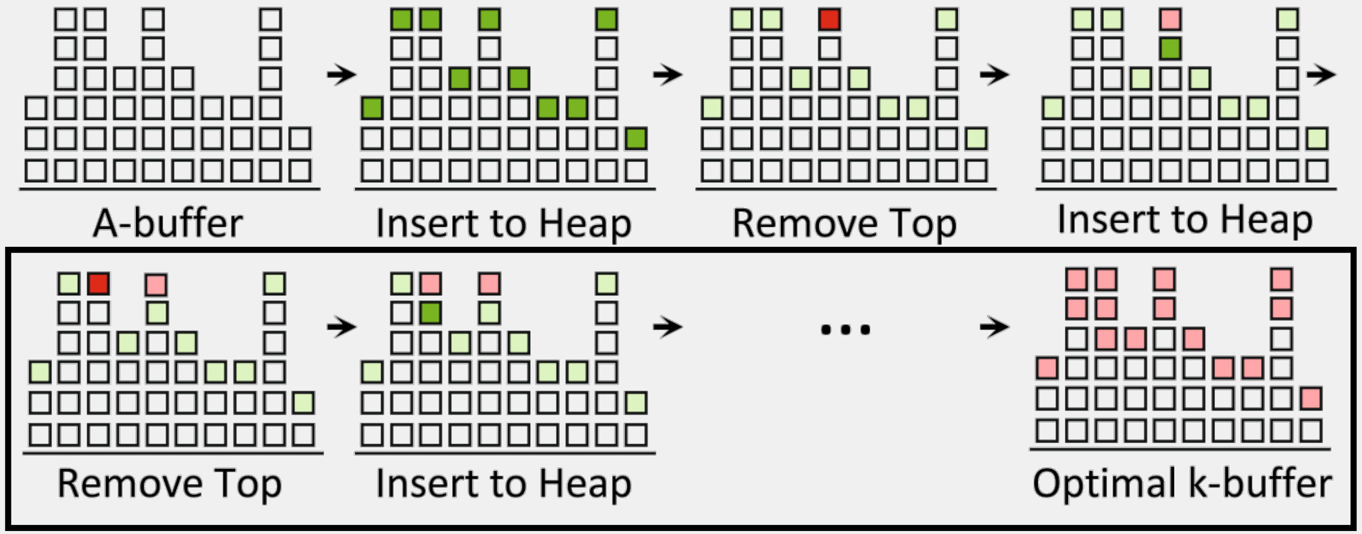
1. Starting from A-Buffer, set the optimal fragment distribution equal to the A-buffer distributions and a desired memory size, and remove at each iteration the farthest from the viewer fragment that causes the minimum perceptual error increase.
2. **For each pixel**, compute the *perceptual error difference* if we **temporarily** remove the farthest fragment, while all other pixels **remain unaltered**, and store it in a **min heap.**

Optimal k-buffer method



1. Starting from A-Buffer, set the optimal fragment distribution equal to the A-buffer distributions and a desired memory size, and remove at each iteration the farthest from the viewer fragment that causes the minimum perceptual error increase.
2. For each pixel, compute the perceptual error difference if we temporarily remove the farthest fragment, while all other pixels remain unaltered, and store it in a min heap.
3. **Remove** the fragment that contributes the least (top of the heap) from the **optimal fragment distribution** and **repeat** [2] only for this pixel.

Optimal k-buffer method



1. Starting from A-Buffer, set the optimal fragment distribution equal to the A-buffer distributions and a desired memory size, and remove at each iteration the farthest from the viewer fragment that causes the minimum perceptual error increase.
2. For each pixel, compute the perceptual error difference if we temporarily remove the farthest fragment, while all other pixels remain unaltered, and store it in a min heap.
3. Remove the fragment that contributes the least (top of the heap) from the optimal fragment distribution and repeat [2] only for this pixel.
4. **Repeat** [3] **until** the desired **memory** size is **reached.**

Experimental Evaluation

- Our method **(DKB)** is compared with **Fixed k-buffer (FKB)** and **Variable k-buffer (VKB)** simulating **Hybrid Transparency**

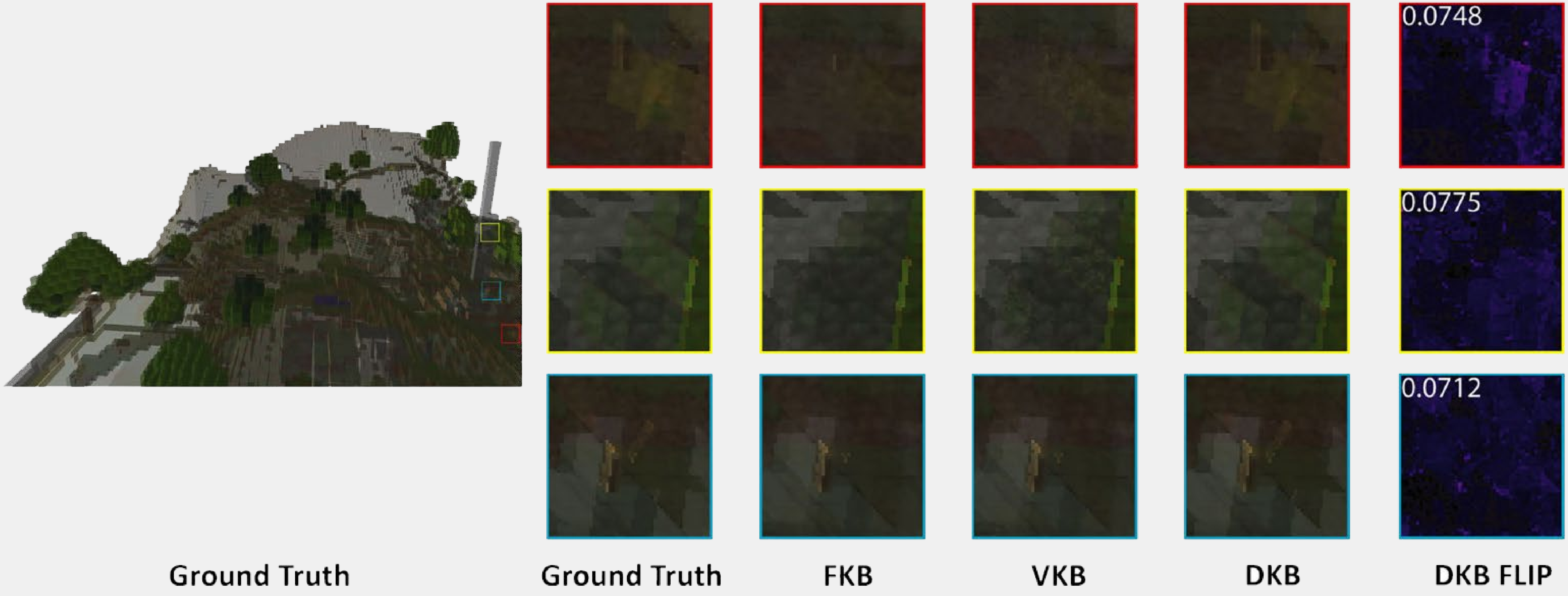
- Different testing scenarios of
  - Varying depth complexity: **30-120 fragments**
  - Memory budgets: **20 & 40 MB**
  - Error metrics : MSE & FLIP [ANA*20]

- **1430 × 960 viewport** on an NVIDIA RTX 2080 Super

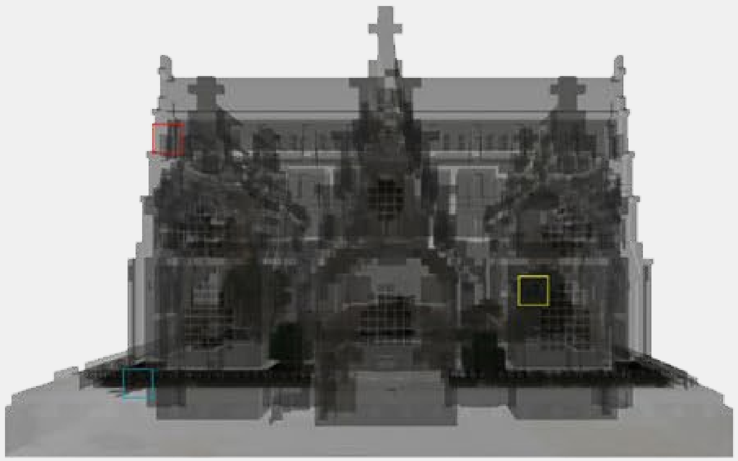
- OpenGL 4.6

[ANA*20] Andersson, et al.: FLIP: A difference evaluator for alternating images, I3D 2020.

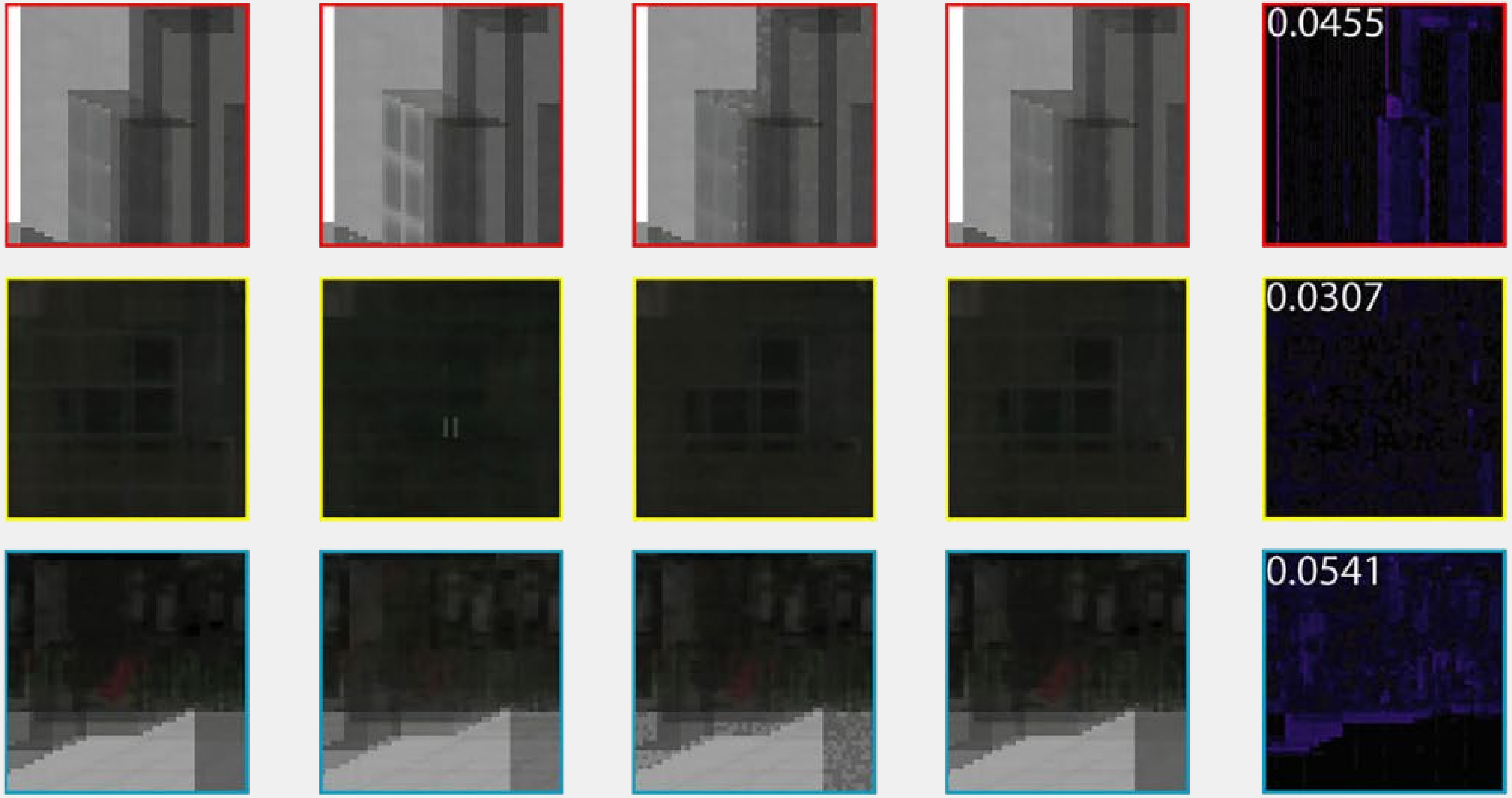
Qualitative Results – Lost Empire (20MB)



Qualitative Results – Rungholt (20MB)



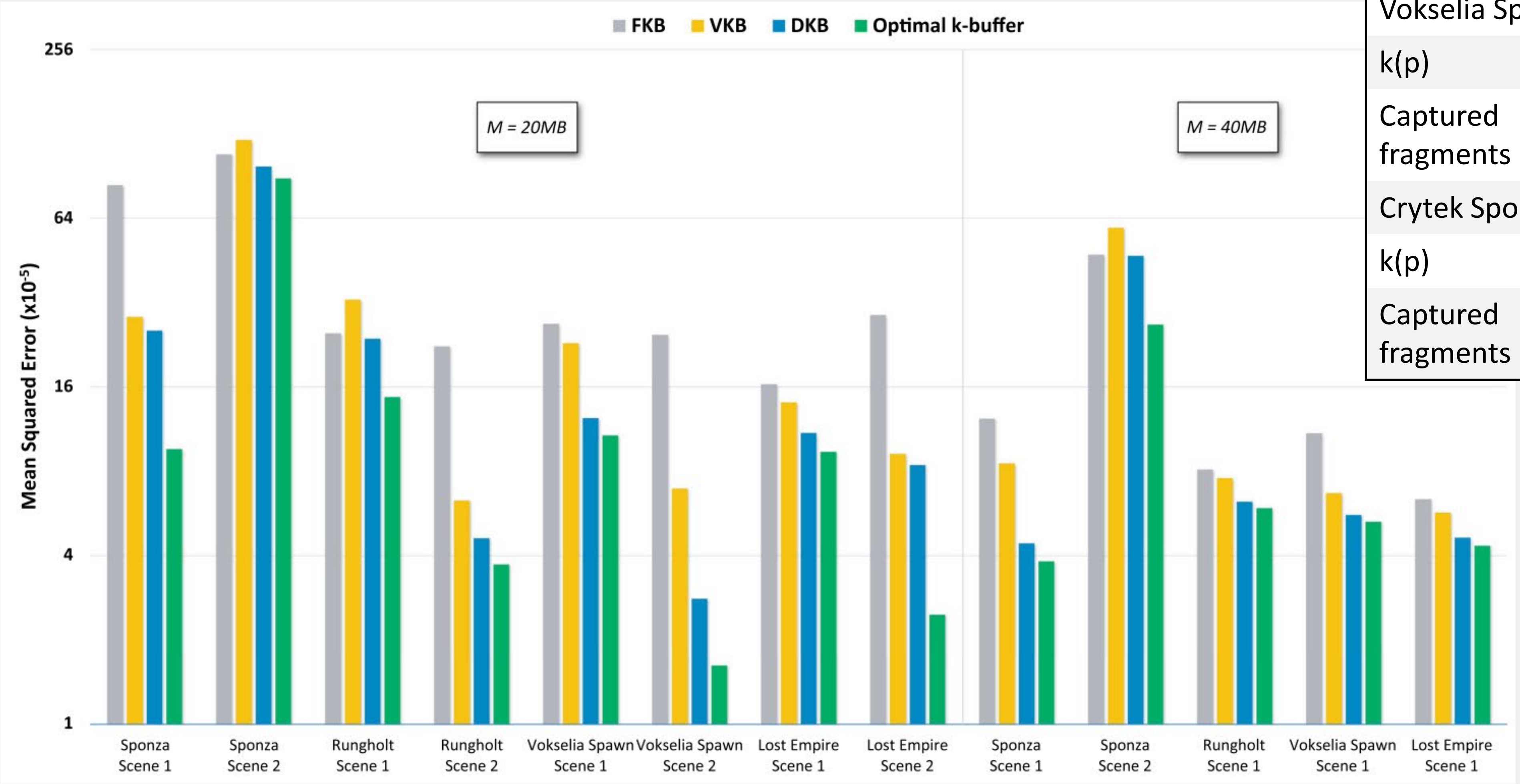
Ground Truth



Ground Truth FKB VKB DKB DKB FLIP

Qualitative Results – Video

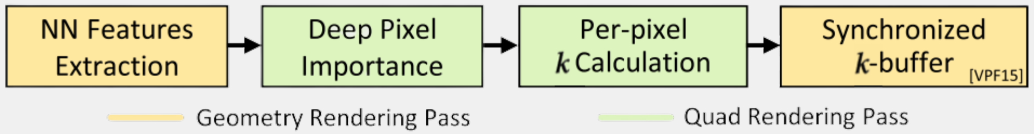
Qualitative Results – Varying memory



Method	**FKB**	**VKB**	**DKB**
Vokselia Spawn, M = **20MB**			
k(p)	2	1-9	1-5
Captured fragments	*24%*	*40%*	*40%*
Crytek Sponza, M = **40MB**			
k(p)	4	2-25	4-25
Captured fragments	*39%*	*83%*	*83%*

Performance Analysis

- **Minor overhead** over VKB
- **Depends on NN inference time**
  - Can be further reduced by simplifying NN
  - Depends on number of transparent pixels



Method	DKB	VKB
Vokselia Spawn, M = ***20MB***		
Feature extraction	1.95	0.9
Importance computation	1.48	0.45
Synchronized k-buffer	18.97	18.97
Total time (ms)	**22.40**	**20.32**
Crytek Sponza, M = **40MB**		
Feature extraction	1.6	0.45
Importance computation	3.25	0.39
Synchronized k-buffer	10.38	10.38
Total time (ms)	**15.25**	**11.22**

Conclusions & Future Work

- The **first deep learning multifragment rendering** method
  - Improves Variable k-Buffer quality, with a minor overhead
  - Distributes fragment storage to more important pixels
  - Uses a simple deep learning mechanism
  - Relies on a novel backward greedy algorithm for optimal fragment distribution

- **Future directions**
  - Different deep learning model architectures (CNN)
  - Different effects (Ambient Occlusion, Global Illumination, Shadows)
  - Game engine integration

Acknowledgements

3D SCENES:

- Lost Empire, Vokselia Spawn, Rungholt and Crytek Sponza were downloaded from Morgan McGuire's Computer Graphics Archive [MG17].

FUNDING:

- This research was supported by project "Dioni: Computing Infrastructure for Big-Data Processing and Analysis" (MIS No. 5047222) co-funded by European Union (ERDF) and Greece through Operational Program "Competitiveness, Entrepreneurship and Innovation," NSRF 2014-2020

- This work has been co-financed by the European Union (European Regional Development Fund- ERDF) and Greek national funds through the Interreg Greece Albania 2014-2020 Program (project VirtuaLand)

Thank you for your attention!

Questions ?

European Union
European Regional Development Fund
HELLENIC REPUBLIC
MINISTRY OF DEVELOPMENT AND INVESTMENTS
SPECIAL SECRETARIAT FOR ERDF & CF PROGRAMMES
MANAGING AUTHORITY OF EPAnEK
EPAnEK 2014-2020
OPERATIONAL PROGRAMME
COMPETITIVENESS ENTREPRENEURSHIP INNOVATION
ΕΣΠΑ 2014-2020
ανάπτυξη - εργασία - αλληλεγγύη
Partnership Agreement 2014 - 2020
Co-financed by Greece and the European Union

The Project is co-funded by the European Union
and by National Funds of Greece & Albania

Interreg - IPA CBC
Greece - Albania
EUROPEAN UNION