A Multiview and Multilayer Approach for Interactive Ray Tracing Demo Documentation

Figure 1: Resulting images of the scenes available in this demo, as they have been rendered with MMRT.

Introduction

This demo is a prototype implementation of the paper *A Multiview and Multilayer Approach for Interactive Ray Tracing* (MMRT) [7], a generic rasterization-based ray tracing framework capable to support approximate solutions to global illumination algorithms, such as path tracing and ambient occlusion, in complex and fully dynamic environments, without the need for precomputations or the maintenance of additional spatial data structures.

The current demo is implemented using OpenGL 4.4 in our custom graphics engine, XEngine [6] (an older version is available here) and has been tested on NVIDIA GPUs with Maxwell architecture or newer. It encompasses a more robust implementation along with several performance and quality improvements compared to the original paper (e.g., better memory management, sampling/lighting calculations, etc.).

For any suggestions regarding the code, e.g., bug fixes, performance improvements, optimizations, etc., or this document, e-mail me at kvardis@aueb.gr, kvardis@hotmail.com, or message me on my Twitter account @kvarcg.

Fast Details about the Demo

The main purpose of this demo is provide an introduction to multi-layer fragment-based image-space ray tracing and how it operates under various conditions. It encompasses a much larger degree of functionality than a product-ready version should have since most of its internal parameters can be altered, e.g., linear/hierarchical tracing, depth subdivisions, ray types, etc. As such, the underlying implementation is unavoidably "heavier" than it should be in terms of both performance and memory, since it must manage various additional buffers and perform extra computations for all its visualization and testing purposes.

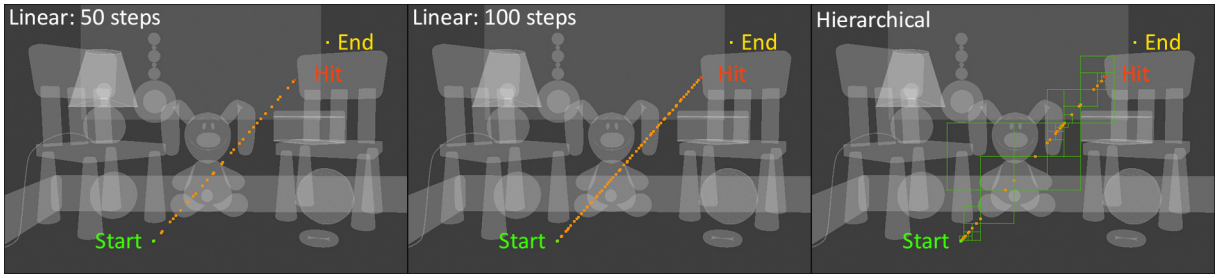

Figure 2: Linear tracing allows for approximate but performance-controlled ray intersections (Left-Middle), while hierarchical tracing offers much more accurate results. Rays travel from left to right.

The resulting images are those of a raw path tracing implementation based on simple BSDF importance sampling or traditional object-based Ambient Occlusion (see Figure 1). There is no filtering applied to the final result. The demonstrated scenes were designed with real-case scenarios in mind and with no regard as to any "optimizations" that improve performance on certain cases, such as on the number of primitives, the placement of the objects in the virtual world, the number and type of materials and/or textures they contain.

Fast Details about MMRT

MMRT relies entirely on the rasterization pipeline to store fragment information for the entire scene on a cube-mapped A-buffer structure, in the form of a view-based (single- or multi-view) perspective irregular grid. During tracing, it marches in image-space and through depth layers to capture both near and distant information for global illumination computations.

The construction stage is responsible for generating and storing fragments in multiple layers/views using a decoupled data storage representation, depth subdivisions and per-pixel double linked lists. Essentially, the underlying acceleration data structure (ADS) is a view-based (single or multi-view) perspective irregular grid. The construction process involves two geometry passes, one for capturing the depth extents of each pixel (stored in a *depth bounds texture*) and one for concurrently storing (in an unsorted order) each fragment to an associated depth subinterval that is represented by a double linked-list, and one final screen-space reorder pass that sorts the stored fragments and assigning the head/tail pointers to the lists. The storage data is split into two units: an *ID* buffer, containing only construction/tracing data, (i.e. depth and head/tail pointers), for sorting and tracing and a *Data* buffer, holding shading information that is only used after any ray-fragment intersections have occurred. Optionally, a (min-max) mipmapped representation of the depth bounds texture can be also computed to support hierarchical screen-space ray tracing.

Once the multi-layer cubemap construction has been completed, image-space ray tracing is performed by (i) iteratively sampling pixel locations in image-space, and (ii) querying the ADS until a ray-fragment collision is detected. As shown in the paper (Section 4.3), naive image- and depth-space (i.e., multi-fragment) ray tracing is very inefficient. MMRT significantly improves the traversal performance by: (i) employing bidirectional traversal through the use of the double linked lists, (ii) performing empty-space skipping through hierarchical or (the more approximate) linear ray tracing in image-space (see Figure 2) and (iii), skipping depth bins or even entire pixels that are not crossed by a ray (see Figure 3). Once a set of appropriate bins has been found, ray-fragment intersection takes place to identify any potential hits.

MMRT supports two ways of capturing the scene and performing ray-tracing: (i) a multi-view structure, capable of capturing the entire scene at interactive rates, and (ii) a single-view structure, able to maintain real-time frame rates while providing global illumination results restricted to the geometry within the view.

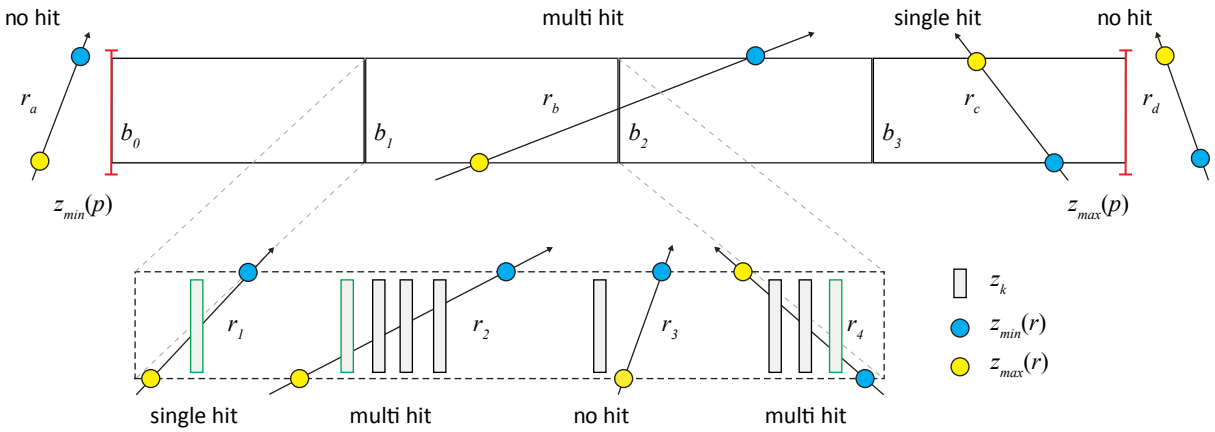

Figure 3: Different ray-bucket (top) and ray-fragment (bottom) hit cases. Rays passing outside the pixel/bucket boundaries can efficiently skip fragment traversal ($r_a, r_d/r_b, r_c$). Rays crossing multiple fragments (r_2, r_4) use their direction to determine the hit.

Demo Details

Requirements

The demo requirements are:

- Windows 64-bit operating system
- OpenGL 4.4
- Visual C++ 2015 x64 Update 3 redistributable (in the redist folder)

To Execute The Demo

The demo contains 5 different scenes of arbitrary complexity. To run a scene simply double-click on its associated bat file. For example:

- *"run_bunny_day_1024_512_full.bat"* executes the *bunny_day* scene at 1024×512 resolution using the "full" MMRT version, i.e., the multi-view version.
- *"run_bunny_day_1024_512_single.bat"* executes the *bunny_day* scene at 1024×512 resolution using the single-view implementation of MMRT.

Notes on First Time Running the Demo

XEngine supports dynamic object loading and a binary representation (*.sbin files) of .obj files to increase responsiveness and improve loading times. The binary files are generated the first time an .obj file is loaded and a warning message will show up indicating that the generation will take place. The scene however (even an empty one) will be functional right from the beginning and will show each object as soon as it becomes ready for rendering.

To disable binary file functionality, set the XML attribute *binary_loading* of the *world* node at the top of the corresponding scene file to *false*. To remove those files at any time, simply delete all *.sbin files in the *EngineData* folder. Finally, to disable dynamic loading, set the *multithreaded_loading* of the *world* node to *false*.

To Run Your Own Scenes

- Create a new folder containing your .obj files in the EngineData folder.
- Create a new .scene file, by copying the *template_mmrt_<views>* files and edit the parameters in the new scene file using a text editor.
- Edit the following parameters in the .mtl files of your obj to support XEngine's internal material system:
  - There is no notion of specular color and *Ks* is used to represent scalar reflectivity. Therefore for a reflectivity of 0.1 use *Ks 0.1 0.1 0.1*.
  - Use *metallic*, in the range of [0-1], to set a material as metallic.
- Create a new .bat file containing the following:

```
start Demo.exe <your_scene>.scene <width> <height> 0 <"Window Title">
exit
```

- Run the new file.

Current Limitations

The current implementation is limited in the following:

- The cubemap version of this demo uses 7 views instead of 6. The additional view is essentially the camera view that is kept mainly to allow the support of the single-view version in the same implementation.
- The background color is not used for rays exiting the view since sparse ray sampling can cause rays to travel within closed walls and requires special handling. Therefore, the outdoor scenes appear darker.
- There is an issue with the current hierarchical implementation that may cause randomly an unnecessary slowdown during user navigation.
- Only diffuse/glossy/specular/emissive materials are supported (no transmission).
- Only one point spotlight source is supported.

General Tips for Best Performance/Memory balance

The MMRT parameters that affect its performance/memory requirements are: sampling type, number of samples/view and depth subdivisions.

- Sampling type and samples per view are provided for best performance/quality balance on recent GPUs. These parameters can be altered by changing the XML attribute *samples_per_ray* in the associated scene file. A value of -2 denotes hierarchical marching, -1 linear conservative marching (i.e., use as many samples as needed to traverse the view) and any positive value the number of samples per view.
- Depth subdivisions should be between 4 and 20. This can be altered by changing the XML attribute *buckets* of the *tracing* element in the associated scene file. Increasing the depth subdivisions also increases the memory requirements.

Troubleshooting

- Check the log file in the Logs folder for any *Warning* or *Error* messages.
- On slow GPUs, there may be the case where construction times are extremely high (>100ms) due to very large fragment generation and/or sorting becoming a computational bottleneck. In this case try a lower resolution version or the single-view variation of the scene.
- If tracing is very slow on your GPU, try reducing the sample/view number either by using the key binding or by changing the XML attribute *samples_per_ray* in the associated scene.

Key Bindings

An extensive list of key bindings are provided for information and testing purposes (see also Figure 4):

Tracing Settings	
M:	Enable/Disable progressive rendering
B:	Changes Application mode between Path Tracing and Ambient Occlusion
3/4:	Increase/Decrease number of light bounces, e.g., 1 is direct lighting, 2 is direct + 1 indirect bounce, etc.
F:	Enable/Disable Depth of Field
C/V:	Change the aperture size to increase/decrease Depth of Field
Z/X:	Increase/Decrease the focus distance, i.e., where the objects appear sharp
Testing Settings	
G:	Changes the View mode, i.e. the number of views between single and cubemap
H:	Changes the Layer mode, i.e. the number of captured layers between single and all (multi-layer mode) (single/all)
5/6:	Increase/Decrease number of depth bins. i.e., the number of per-pixel depth subdivisions
7/8:	Increase/Decrease the fragment's thickness for intersection tests
J	Changes the Traversal mode between linear and hierarchical tracing
9/0:	Increase/Decrease the number of ray samples per view for linear tracing and the highest mip-level (i.e., low-detailed) one for hierarchical tracing. These can be used in conjuction with the Ray Preview mode to visualize how the rays traverse the scene.
[/]:	Increases/Decreases the fragment layer shown in Preview mode in order to visualize how fragments are captured in a multi-fragment buffer.
Y	Enables Disables Fragment Heatmap visualization, i.e., display the number of layers/faces the rays intersect with. This is used in conjunction with the View/Layer modes (G, H) to illustrate the importance of multi-view/layer tracing compared to their single-view/layer alternatives.
-:	Enable/Disable Ray Preview mode. This causes the ray of the central pixel to be written to a debug ray texture and previewed in Preview mode.
=:	Changes the rays to reflect different materials. Ray types cycle between: Default (no change), diffuse, glossy (roughness=0.1), specular and visibility.
Engine Settings	
T:	Enable/Disable GPU timers
F8:	Show/Hide screen text
P:	Preview internal buffers
1/2:	Prev/Next preview buffer
F5:	Screenshot capture of the current preview buffer
F6:	Export camera and user details, such as position, target, direction, near and far field, in the Log file
R:	Reloads the shaders
ESC:	Exit the application
Camera Movement	
Arrow keys:	User movement on camera UW(XZ) axis
PgUp/PgDn:	User movement on camera V(Y) axis
Mouse(Left Button):	Rotation on UV(XY) axis

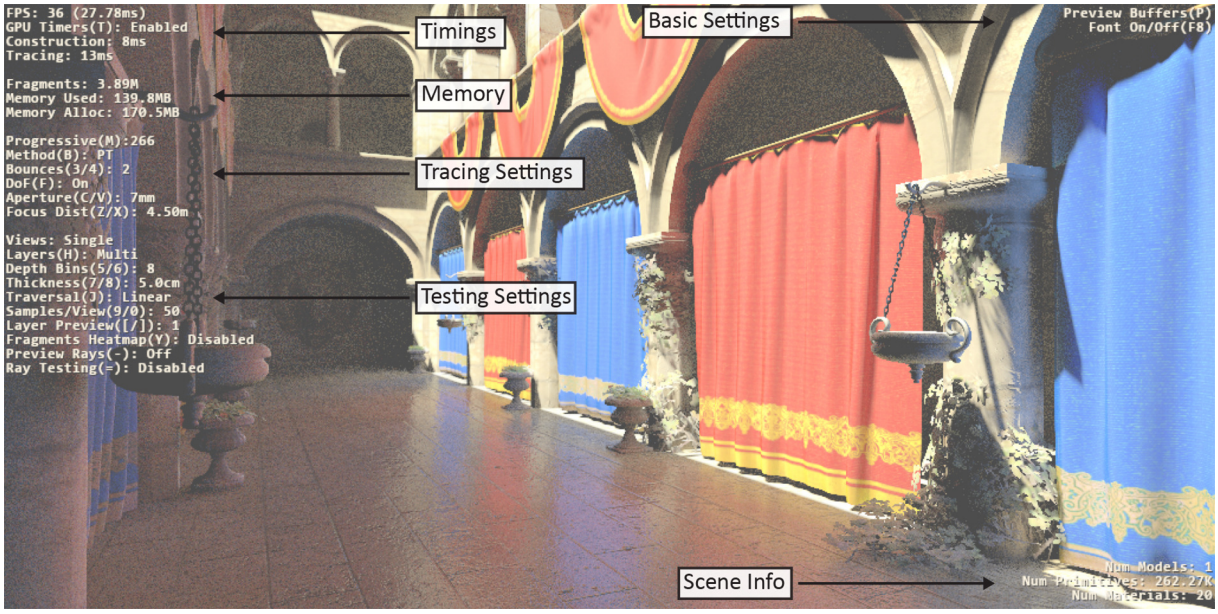

Figure 4: The screen text displays information relevant to memory, performance and tracing as well as various visualization/testing settings.

General Notes Regarding Quality/Performance/Memory/UI

- The thickness parameter assumes each sample is a frustum-shaped voxel of non-zero thickness. It is commonly present in fragment-based tracing methods to mitigate inaccurate ray-fragment intersections (hits or misses) due to geometry being sparsely sampled in the rasterization pipeline. While reasonably effective, this parameter requires manual adjustment, can introduce view-dependencies and has the effect of introducing holes or extrusions (for small and large values respectively) on the tested objects, errors that are mostly visible when high frequency phenomena are present. This can be remedied by employing more accurate and computationally intensive primitive-based ray tracing techniques (see the discussion on the corresponding paper [5] and its demo on the Computer Graphics Group's website of the Athens University of Economics and Business [2]).
- The actual memory allocations are higher than needed (see Memory in Figure 4) (i) to accommodate both single-/multi-view implementations (e.g., 7 views instead of 6), (ii) to provide a testing/visualization environment (see Test Settings in Figure 4) and, (iii) due to functionality limitations in OpenGL. These have also an impact on the performance of the current implementation.
- The current implementation of hierarchical tracing currently supports resolutions in powers of two (e.g., 512×512, 1024×512, 1024×1024, etc.). Window sizes are adjusted this way and therefore, resize is not available. Linear tracing, however, is not limited by this and is the only tracing option in the 1920×1080 scenes.
- Use GPU timers to see the exact construction/tracing times. Since all displayed timers are measured synchronously, disable them for better performance.
- Switching between single/cubemap view is not available in the single-view demo, as it requires reallocation of all the internal buffers.
- Depth bins cannot be larger than the initial state, as it also requires reallocation of all the internal buffers.
- Ray Preview and Fragment Heatmap modes should be used for visualization purposes only as they impose a noticeable performance overhead during tracing due to the extra write operations in the ray preview buffer.

- The different ray types are demonstrated on the *indirect* paths as direct shading is very fast. They should only be used for benchmarking and visualization purposes. When visibility rays are selected light bounces is set to 2 (1 indirect).
- Maximum number of light bounces is 3.

GLSL Source Code Reference

Example Source Code

Example source code (i.e., no path tracing or ambient occlusion implementation) is contained in the *Example Shader Code* folder. This is the sample shader code that was provided as supplemental material in the original paper.

ADS Construction Stage

The shader source code for the ADS construction stage is located at *Shaders\Tracing\FragmentConstruction* and contains the following files:

- *MMRT_DepthBoundsCompute*: Geometry pass to computes the min/max depth bounds for each pixel.
- *MMRT_DepthBoundsMipMap*: Computes a mipped representation of the depth bounds, used for tiling and hierarchical tracing.
- *MMRT_Store*: Geometry pass to construct the ID and Data buffers. Fragments are stored in an unsorted order.
- *MMRT_Reorder*: Screen-space pass used for sorts the fragments onto their corresponding buckets, fixing the head/tail pointers and previewing the MMRT ADS.

Traversal Stage

The code is located at *Shaders\Tracing\FragmentTracing* and contains the following files:

- *MMRT_PT*: A basic path tracing implementation.
- *MMRT_AO*: A basic ambiento occlusion implementation.
- *MMRT_Preview*: Screen-space pass used for (i) combining an ADS view with the ray preview, and (ii) viewing all ADS views along with the final result in the same preview window.

Helper GLSL Files

The shader source code for all included files is located at *Shaders\Include\MMRT* and contains the following files:

- *MMRT_data_structs.glsl*: Data structs definitions.
- *MMRT_basic_lib.glsl*: Basic functionality, e.g., random number generators, normal compression routines, etc.
- *MMRT_sort.glsl*: Fragment sorting functions.
- *MMRT_vertex.glsl*: Vertex data construction.
- *MMRT_lighting.glsl*: Path tracing illumination and shadow mapping routines.

- *MMRT_tracing_hiz.glsl*: Single and multi-view hierarchical screen-space ray tracing.
- *MMRT_tracing_linear.glsl*: Single and multi-view linear screen-space ray tracing.
- *MMRT_abuffer_cubemap_hiz.glsl*: Per-pixel linked-list tracing for hierarchical tracing, depth-based skipping and ray-fragment intersections.
- *MMRT_abuffer_cubemap_linear.glsl*: Per-pixel linked-list tracing for linear tracing, depth-based skipping and ray-fragment intersections.

Content Credits

- The Sponza Atrium model was downloaded from Morgan McGuire's Computer Graphics Archive [1].
- The Smithy model was downloaded from Unity Technologies and is part of the The Blacksmith Environments package [4].
- The high resolution version of the decimated Lucy model (Level1 scene) was obtained from the Stanford University Computer Graphics Laboratory [3].
- The remaining models comprising the Bunny and Level1 scenes were created by Prof. Georgios Papaioannou at the Computer Graphics Group of the Athens University of Economics and Business [2].

References

[1] M. McGuire. *Computer Graphics Archive*. July 2017. URL: https://casual-effects.com/data (Cited on page 8).

[2] G. Papaioannou. *Computer Graphics Group*. URL: http://graphics.cs.aueb.gr/graphics/downloads.html (Cited on pages 6, 8).

[3] Stanford University. *Computer Graphics Laboratory*. URL: http://graphics.stanford.edu/data/3Dscanrep (Cited on page 8).

[4] Unity Technologies. *The Blacksmith Environments*. URL: https://www.assetstore.unity3d.com/en/#!/content/39948 (Cited on page 8).

[5] K. Vardis, A. A. Vasilakis, and G. Papaioannou. *DIRT: Deferred Image-based Ray Tracing*. In: *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. Ed. by U. Assarsson and W. Hunt. The Eurographics Association, 2016. ISBN: 978-3-03868-008-6. DOI: 10.2312/hpg.20161193 (Cited on page 6).

[6] K. Vardis and G. Papaioannou. *XEngine*. URL: http://graphics.cs.aueb.gr/graphics/downloads_tutorials.html (Cited on page 1).

[7] K. Vardis, A. A. Vasilakis, and G. Papaioannou. *A Multiview and Multilayer Approach for Interactive Ray Tracing*. In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '16. Redmond, Washington: ACM, 2016, pp. 171–178. ISBN: 978-1-4503-4043-4. DOI: 10.1145/2856400.2856401 (Cited on page 1).