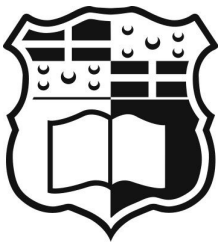
# Real-time GI for dynamic environments using temporal filtering

**Kevin Tonna**

**Supervisor(s):** Keith Bugeja and Sandro Spina



**Faculty of ICT**

**University of Malta**

May 2019

*Submitted in partial fulfillment of the requirements for the degree of B.Sc. Computing Science (Hons.)*

**Abstract:**

Real-time rendering applications have traditionally shunned from implementing fully dynamic global illumination solutions due to their high computational cost, despite the added value in terms of realism and image quality the latter can afford. However, with the recent advent of rendering hardware accelerating ray tracing-based methods, new avenues of research have opened up that investigate the use of offline global illumination rendering techniques for real-time purposes. In this dissertation, we propose a method for interactive global illumination that reformulates IGI, a traditionally CPU-based rendering approach, and augments the derived solution with information from previously rendered frames. The proposed method is fully dynamic, in that no pre-computation is required and scene objects such as light sources, geometry and materials may be updated without any performance penalty. We evaluate the method's performance in both static and dynamic scenes, and compare the resulting image quality against ground truth images generated from an unbiased offline renderer. Results are promising and show that the method may be viably used for interactive applications, without degrading the quality of the output.

# Acknowledgements

I would like to thank my supervisors Keith Bugeja and Sandro Spina for their time and advice, as well as my friends and family for their encouragement and support.

# Contents

# Real-time GI for dynamic environments using temporal filtering

Kevin Tonna*

**Supervised by:** Keith Bugeja and Sandro Spina

May 2019

## 1 Introduction

Global illumination is the combination of direct and indirect illumination. Direct illumination involves computing light rays directly from a surface point to a light source. Indirect illumination light rays bounce multiple times and contribute to light phenomena such as refractions and reflections. Contrary to direct illumination, indirect illumination is complex to calculate due to having to consider all the surfaces in the scene multiple times for illuminating one surface. This is why real-time applications generally stick to only using direct illumination.

Global Illumination is crucial for realistic graphics. While this has been achieved in offline graphics, it is non-trivial to use global illumination for real-time applications due to its complexity. Algorithms which take advantage from recent advances in rendering hardware, such as the 'reflections' ray-tracing demo [1] showed that the reflection phenomena of real-time GI is achievable, but the hardware behind it is a far stretch from consumer hardware. Therefore, research is still ongoing on rendering algorithms which boost the current performance possible at slight costs.

Illumination exhibits temporal coherence. This allows the opportunity to use temporal filtering to reuse illumination information computed on previous frames when drawing the current frame, cutting down on costs. Using re-projection, previous frames can be repurposed to accommodate any scene changes from one frame to the next. For example, if the camera in a scene is rotated upside down from one frame to the next, instead of drawing the scene from scratch, the previous frame can be rotated to fit the changes in the scene.

*Submitted in partial fulfillment of the requirements for the degree of B.Sc. Computing Science (Hons.).

This dissertation aims to improve on instant radiosity GI algorithms for diffuse surfaces using hardware accelerated ray-primitive intersections via a method that makes use of temporal filtering to alleviate costs associated with instant radiosity. An interactive hardware accelerated instant radiosity application will be implemented on to which the method will be applied. This will allow evaluation of the method's performance benefits and any quality deficits.

# 2 Background

In this section some topics will be explained to aid the reader in understanding the rest of the dissertation. These topics are : projections and their involvement in scene transformation, lighting and the components involved, and acceleration structures and their performance benefits.

## 2.1 Projections

3D models are represented as vertices connected to each other with edges. The edges are then connected together to make up faces. Models are stored independent of their position within a scene, typically the origin represents the center of the model or some other point of importance from which the model is rotated and scaled from. Each vertex is multiplied by a model transform that represents the transformation necessary to move the vertices to their place in the scene. A transform is a matrix that when applied to a vertex, the vertex can be translated, rotated and scaled. By multiplying transforms with each other, they can be combined to make more complex operations such as returning a vertex in a different co-ordinate space. After moving vertices from model space to world space, the vertices need to be moved into camera space. Using another transform the scene is transformed such that the visible part of the scene fits inside a unit cube. The camera transform also applies the camera projection such as orthographic projection, where parallel lines are kept parallel and perspective projection, where objects appear larger the closer they are to the camera. At this stage for performance reasons, any vertices outside the unit cube are clipped so that only the visible faces are considered in the next stage. On GPUs this is done by hardware and thus is not programmable. In the last stage, from clip space to screen space, rendering goes from a per vertex basis to per pixel. This is where faces are filled in with colour according to lighting and face properties.

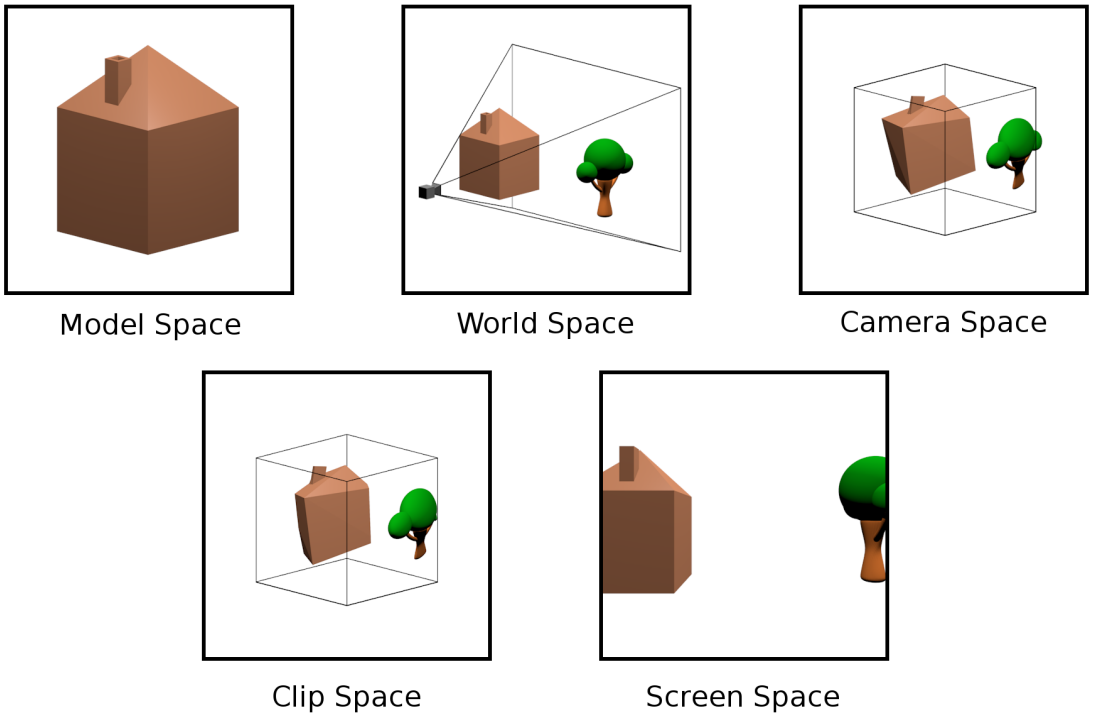



Figure 1: Transformations from Model Space to Screen Space

## 2.2 Lighting

Light sources refer to objects in a scene that emit light. Diffuse surfaces reflect light equally in all directions, making diffuse lighting independent of the angle between the surface normal and the direction of sight. Instead, diffuse lighting depends on the angle between the surface normal and light source. Specular surfaces reflect light opposite the angle of incidence, thus specular lighting is dependant on the angle between the reflected light ray and the direction of sight. Combining these two allows for a range of materials to be represented.

Light loses energy as it travels through a medium. When calculating lighting, the energy is attenuated by distance travelled by the light ray. This is why direct lighting makes up the majority of the lighting in a scene, as light rays lose energy with each bounce.

Shadows are important for realistic lighting. For direct lighting, shadow is usually calculated by rasterising the scene from the point of view of the light source [2]. The result can be looked up while rendering direct lighting to check if a surface is in shadow or not.

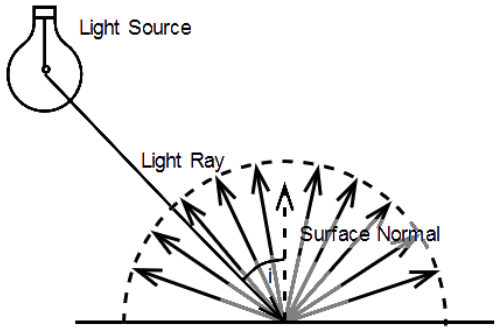



Figure 2: Diffuse Lighting

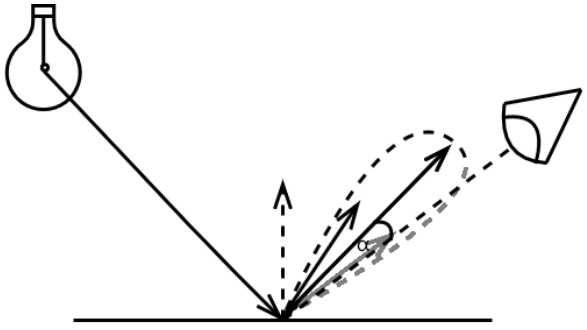


Figure 3: Specular Lighting

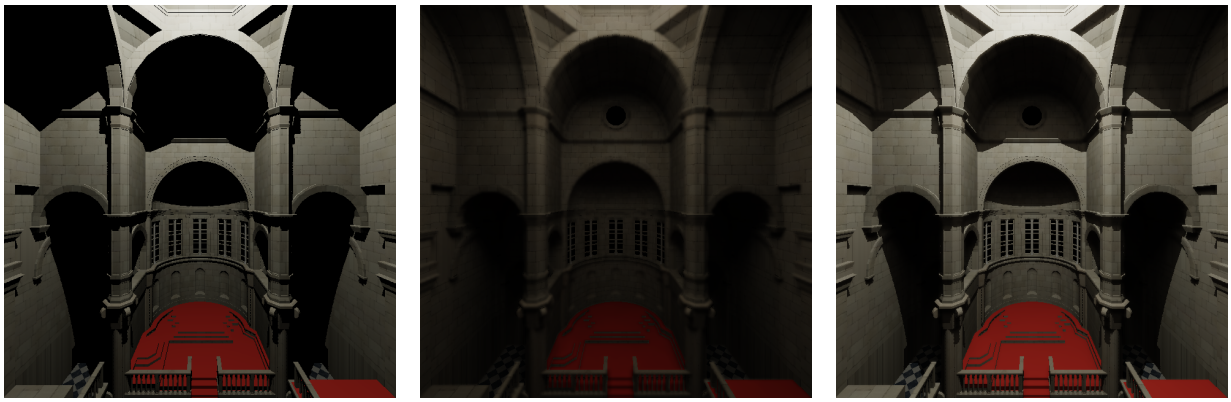


Figure 4: Direct, Indirect and Global Illumination

This would be too taxing for indirect lighting as the scene would have to be rasterised much more times each frame.

## 2.3 Acceleration Structures

Bounding volumes estimate complex objects to make operations more efficient, such as ray intersection tests (necessary for shooting VPLs from the light source). For example if a ray does not intersect a box surrounding a complex object, there is no need to perform heavier computations to check if the ray intersects the complex object. Kay and Kajiya's Ray tracing complex scenes [3] introduce Bounding Volume Hierarchies (BVH) that represent 3D scenes using bounding volumes to enable the benefits described above.

A BVH may take time to build before it is able to be used. While this is fine for static scenes, as it only needs to be built once, building the BVH from scratch each frame for dynamic scenes can become taxing. Segovia's Memory Efficient Ray Tracing with Hierarchical Mesh Quantization [4] describes an improvement on BVH useful for dynamic

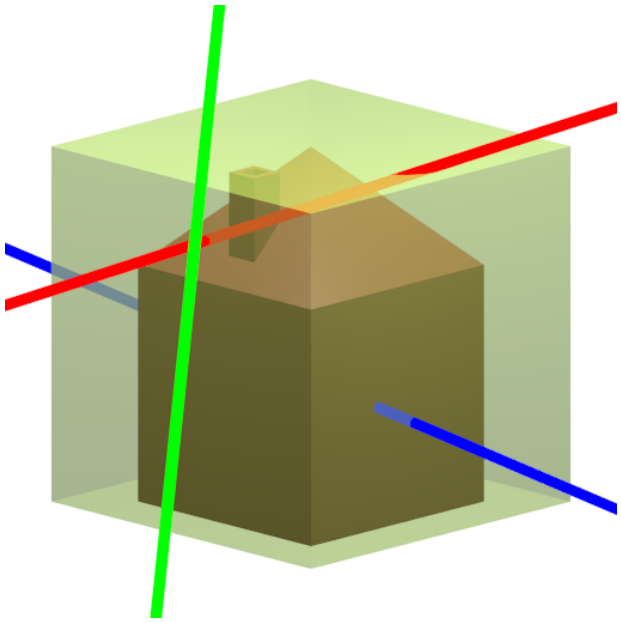


Figure 5: Bounding Volume Example, only the simple test is run on the green ray while both the simple and complex test are run on the red and blue rays. Only the blue ray returns a valid hit.

scenes. This is achieved by splitting the BVH into two-levels, keeping the top of the tree uncompressed and the rest compressed. Compressing a BVH is required due to increasingly large scenes being used nowadays. By keeping the top uncompressed, this prevents the performance disadvantages of compression for the majority of traversals in a BVH.

An octree serves the same purpose of a BVH, to efficiently represent a 3D scene, except in a different manner. An octree is a tree where each node represents a cube in 3D space, and each internal node always has 8 children, representing the 8 corners of a cube. Each leaf node holds a reference to the objects inside its cube. This makes checking for collisions easier by returning only the objects inside the space where a collision is being checked for, reducing the number of objects on which complex collision methods need to be executed. However this can be unsuitable for dynamic scenes as it would require constant sorting of the octree. This involves both static and dynamic parts of the scene since they are represented together in space instead of separate objects. For example if a dynamic object moves from one side of the scene to another, the octree would require searching for all references to the object in its nodes and removing the references if the object no longer present in the nodes' cubes. References also need to be added to other nodes whose cubes are now intersected by the object. In the worst case scenario, leaf nodes which previously

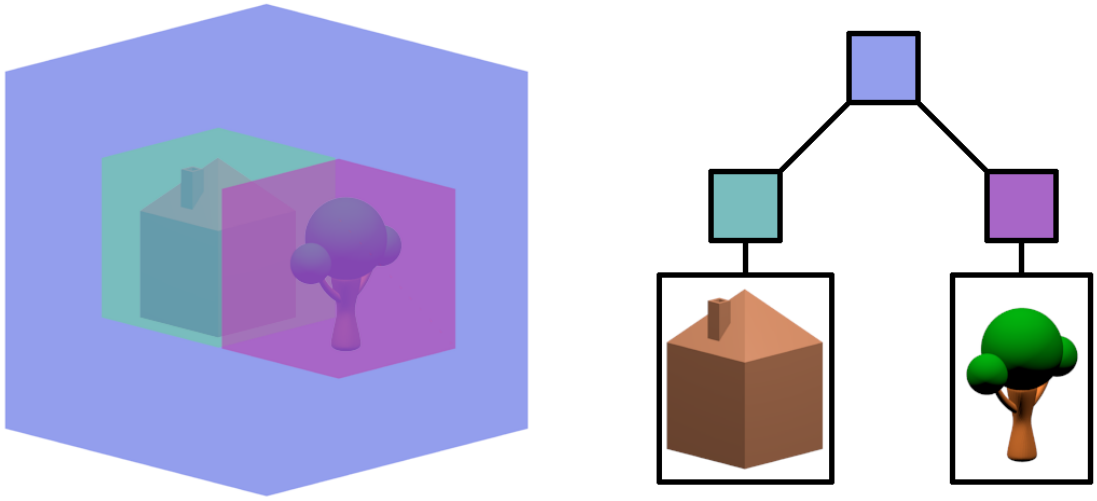


Figure 6: Bounding Volume Hierarchy example

contained the object need to be merged with their parent nodes, while the nodes which now contain the object need to be split in order to continue reaping the performance benefits of an octree.

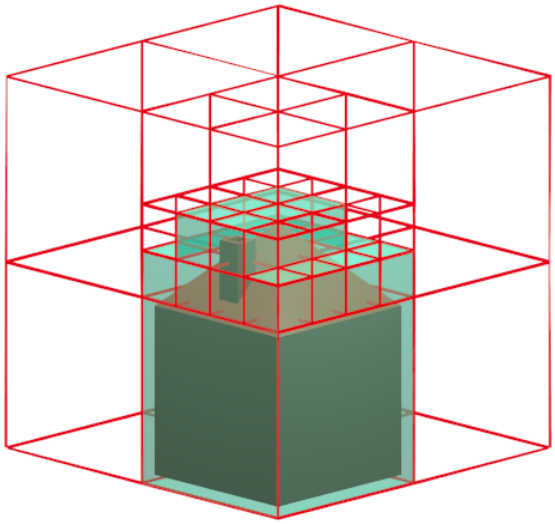


Figure 7: Octree Example, only the nodes with references to the house are shaded.

# 3 Literature Review

Whitted first proposed ray tracing [5] where light bounces are simulated as rays (straight lines) in 3D space. Rays are shot from the camera per pixel into the scene, the rays are refracted and reflected along any translucent or reflective surfaces until a diffuse surface is hit. Another ray is then shot from the diffuse surface to the light source to detect if the surface is in shadow. This method was impressive for the time however its so compute heavy that it took years for it to be viable in offline rendering.

In Distributed Ray Tracing [6], ray tracing is expanded to simulate fuzzy light phenomena without requiring additional rays to be shot, maintaining performance levels. This is achieved by distributing existing rays across functions in the ray tracing pipeline which had been previously approximated as constant. The phenomena simulated by this method are :

- Gloss - by sampling the specular distribution function
- Translucency - by sampling the transmitted ray
- Penumbra - by sampling the solid angle of light sources
- Depth of field - by sampling the camera lens area
- Motion blur - by sampling in time

Kajiya's Rendering Equation [7] generalises lighting algorithms as approximations to the solution of a single equation. This is possible since most lighting algorithms aim to achieve realistic lighting. The rendering equation is as follows:

$$I(x, x') = g(x, x')[\epsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')dx'']$$

Where:

- $I(x, x')$ is the intensity of light from point x' to point x
- $g(x, x')$ describes how the geometry affects the intensity of light
- $\epsilon(x, x')$ is the intensity of emitted light from x' to x
- $\rho(x, x', x'')$ is the intensity of light from x" scattered by x' to x

Attempting to implement the rendering equation in it's entirety would be futile as by definition a surface's lighting is dependent on all surfaces including itself in a recursive manner.

Greenberg's Radiosity [8] algorithm divides surfaces into patches and stores per patch : the energy emitted from the patch, the patches visible and their visibility as a fraction over all directions from the original patch. This means that the entire scene can be calculated independent of the view, making rendering a step of collecting the energy from each patch visible to the camera. Yet for dynamic environments the cost of calculating the patches' information is too high. Specular surfaces are not supported since without having reflected light distribute equally, the visibility of each patch can not be stored as a simple fraction. However this work was later expanded upon to include specular surfaces while keeping the ability to store scene illumination independently of the view [9].

Virtual point lights (VPL) reuse techniques used in direct lighting for indirect lighting. Introduced in Keller's Instant Radiosity [10], VPLs approximate Global Illumination as follows:

1. Quasi-random rays are emitted from the light sources in the scene.

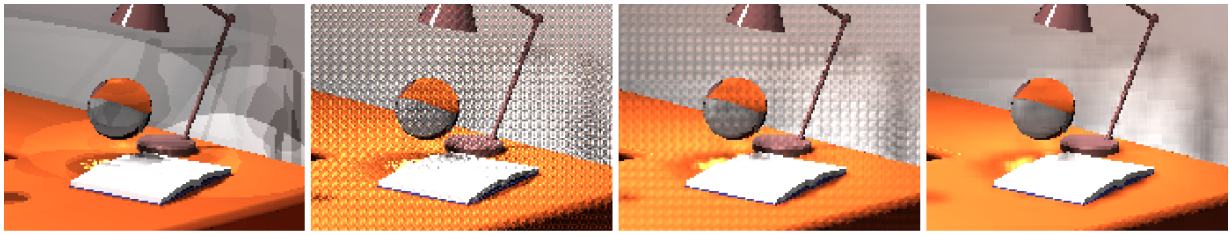
2. At each ray hit, a VPL is placed in which the properties of the ray and the surface are merged together.

3. The scene's lighting is accumulated by drawing the scene for each light source and VPL.

This method reduces the amount of rays shot, since instead of an independent light path per pixel, after the first bounce from the camera to the scene, the light paths are identical. This has the caveat of producing spotlight artefacts where VPLs are hit. This is evident when a low number of VPLs are used.

To better approximate Global Illumination a large amount of VPLs are required. This can be taxing on performance and memory for real-time Global Illumination. Wald's Interactive Global Illumination [11] improves on real-time Instant Radiosity by parallelising the task and only using a sample of VPLs per pixel - interleaved sampling. The use of interleaved sampling leads to a dithered output since each pixel uses different lighting from its neighboring pixels. This can be resolved by smoothing the image. To preserve hard edges, the difference in depth between pixels is used as a smoothing cut-off to keep different surfaces seperate (discontinuity buffer). A Scalable Approach to Interactive Global Illumi-

nation [12] improves upon IGI by removing scalability bottlenecks and adding support for textures, anti-aliasing and tone mapping.



a) no interleaved sampling no discontinuity buffer
b) $5 \times 5$ interleaved sampling no discontinuity buffer
c) $5 \times 5$ interleaved sampling $3 \times 3$ discontinuity buffer
d) $5 \times 5$ interleaved sampling $5 \times 5$ discontinuity buffer

Figure 8: Interleaved Sampling and the Discontinuity Buffer [11]

By only shooting VPLs from light sources, complex situations where the light source is in a different room than the camera can be improperly lit. This is due to VPLs not being placed in the same room as the camera. In Bidirectional Instant Radiosity [13], VPLs are shot from the camera which are then connected to the VPLs shot from the light sources. This improves illumination in scenarios like the one just described.

Lain's Incremental Instant Radiosity [14] reuses VPLs to reduce the amount of intersection tests needed per frame. VPLs are tested and removed if they are in shadow to be reshot. New VPLs are shot to replace the invalid VPLs, some valid VPLs may be replaced for a better VPL distribution. A maximum number of new VPLs shot is kept per frame, This maintains a consistent performance level. Each VPL is given a shadow map to avoid repeating shadow tests, this works well for static scenes, but the shadow maps would need to be recalculated every frame for accurate dynamic scenes. VPL shooting is limited to one hit only however this has been demonstrated to give good results.

Radax's Instant Radiosity for Real-Time Global Illumination [15] mentions Deferred Shading as a possible extension to Instant Radiosity. Deferred Shading refers to gathering geometry data into a G-buffer in one pass (rasterisation) and using the G-buffer to shade the pixels in a second pass (shading). This saves time from shading objects which will be occluded by other objects from the camera's view. Deferred Shading is especially compatible with Instant Radiosity due to needing to draw the scene per VPL each frame, having to rasterise the scene only once is clearly an advantage. The G-buffer just described typically contains information on position, normal, albedo and specularity per pixel.

VPLs are not the only method which approximates GI at interactive rates. The next three algorithms have been shown to achieve real-time frame rates.

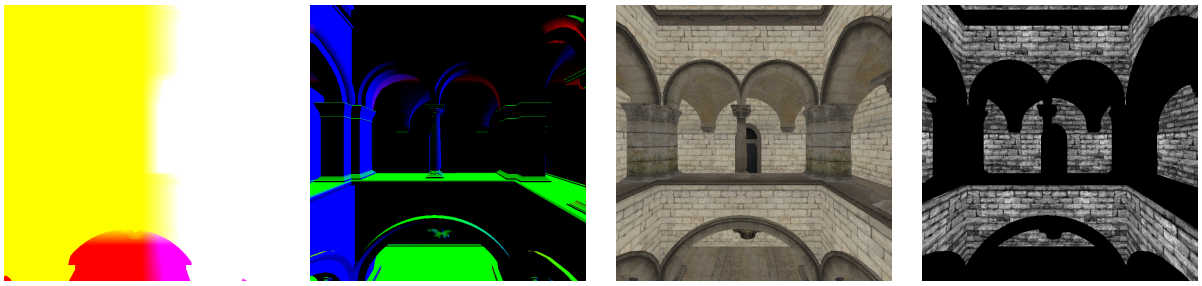


Figure 9: G-buffers : Position, Normal, Albedo, Specularity

In contrast to VPLs, Jensen's Global Illumination Using Photon Maps [16] uses photons. Photons while shot similarly to VPLs to store global illumination in a scene, track incident illumination instead of exitant illumination. This means that when calculating a surface's lighting, only nearby photons need to be considered, an advantage over VPLs. Hardware-Accelerated Global Illumination by Image Space Photon Mapping [17] is an improvement on photon mapping by taking advantage of rasterising hardware. Specifically, the need to individually ray trace photon paths from the light source to their first bounce and from the camera to visible surfaces is eliminated. This is done by rasterising the scene from the camera and light source's point of view and sampling the output as needed during the rendering process.

Cascaded Light Propagation Volumes [18] involve keeping track of 2 volumes, light propagation (LPV) and geometry (GV). These are stored in 3D grids, and record light intensity and blocking potentials respectively. Light intensity direction is defined over the 3 axes using spherical harmonic vectors. The flux for each neighbouring LPV cells' face is calculated as the volume of the SH vector inside the solid angle from the center of the source cell to the face. The new light intensity SH vector is then calculated as the accumulation of the flux from the cell's faces. Shadows are accounted for in the new light intensity by looking up the appropriate GV cells. This process is repeated until the entire grid is filled up. Nested grids are used to provide details close to the camera, while reducing iterations needed by lowering detail with distance. Although this method suffers from light bleeding and missing shadows for objects smaller than the cell size, it allows for dynamic real-time global illumination at a lower detail on consumer hardware.

Interactive Indirect Illumination Using Voxel Cone Tracing [19] voxelises 3D scenes. This only needs to be done once for static objects, allowing for simpler light transport methods at the cost of lower resolution. The voxelised scene is stored as a sparse octree. Rays are grouped together through cone tracing - a complex technique made simple due

to the scene representation. Voxel cone tracing iterates over the length of the cone at steps equal to the resolution of the sparse octree at the current location. Once the indirect illumination has been calculated inside the voxelised scene, the results are sampled in the original scene. Like the previous algorithm, this gives good framerates at the cost of lower detail.

Walter's Interactive Rendering using the Render Cache [20] introduces the concept of storing and reusing the output of the renderer as 3D points. This allows the points to be projected independently of the view they were sampled from. The 3D points' age is tracked to be replaced after an allotted amount of time. New points are sampled instead of entire images from the renderer when replacing old points or lacking enough data in an area of the current view. This concept avoids sampling new points in dense areas by smoothing over the result. These features combined allow the Render Cache to cut costs on any rendering method due to its modular nature.

# 4 Design

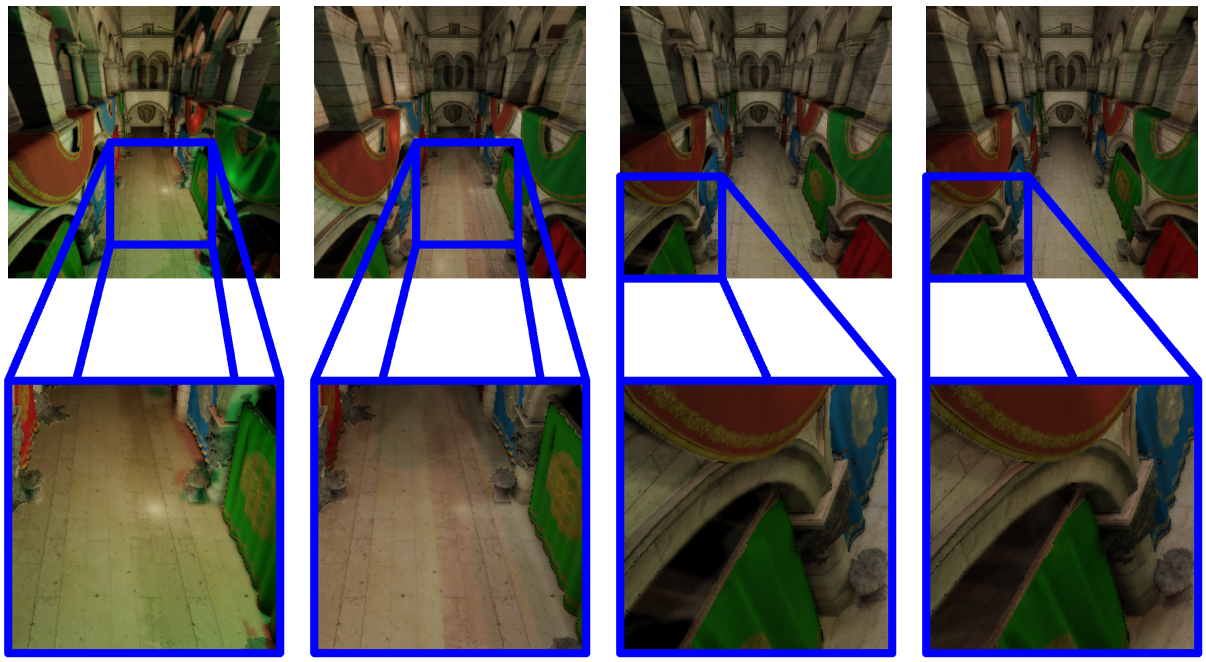


Figure 10: Indirect Lighting and it's artefacts at 8, 32, 128 and 512 VPLs

Instant Radiosity requires a large number of VPLs to produce convincing results. This is because the less VPLs used, the more contribution each VPL has in a scene. As seen in figure 10, at 8 VPLs the location of each VPL is visible due to the bright spotlights next to them, the scene is also not lit uniformly. At 32 VPLs the artefacts from low VPL count are still present, but are less noticeable. At 256 VPLs the VPL's positions are not apparent and the scene is uniformly lit except for harder to reach places such as underneath the arches. This issue is less prominent at 512 VPLs. The system designed, on which temporal filtering is applied to increase the number of simultaneous VPLs possible, will now be explained stage by stage in it's pipeline.

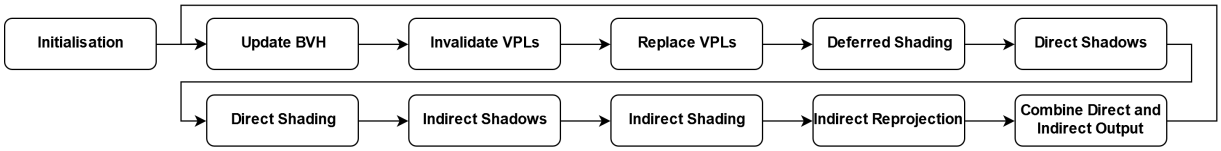



Figure 11: Frame Stages Overview

**Algorithm 1:** Pipeline Stages Pseudocode

```
1  Initialisation()
2  while online do
3  |   UpdateBVH()
4  |   InvalidateVPLs()
5  |   ReplaceVPLs()
6  |   DeferredShading()
7  |   DirectShadows()
8  |   DirectShading()
9  |   IndirectShadows()
10 |   IndirectShading()
11 |   IndirectReprojection()
12 |   CombineDirectAndIndirectLighting()
13 end
```

On initialisation (line 1) the scene is loaded in. All VPLs are set as invalid to be shot later in the pipeline. The BVH is constructed from scratch during initialisation, from then on any changes to the scene are reflected by updating the BVH (line 3). Static and dynamic portions of the scene are distinguishable inside the BVH allowing for faster updates. Invalidating VPLs (line 4) involves running occlusion tests from each VPL to its

parent VPL / light source. A VPL is invalidated if it is found to be in shadow from its parent VPL / light source. If an invalidated VPL has children, they have to be invalidated as well.

**Algorithm 2:** InvalidateVPL() Pseudocode

```
for VPL in VPLs do
    if IsValid(VPL) and IsOccluded(GetParent(VPL, VPLs), VPL) then
        while VPL != null do
            SetValidity(VPL, false)
            VPL = GetChild(VPL, VPLs)
        end
    end
end
```

New VPLs are shot to replace any invalidated VPLs (line 5). This is limited to an arbitrary maximum each frame to prevent performance hits when a large number of VPLs are invalidated at once. VPLs are stored as an array where the VPLs index defines the light path the VPL is part of and the VPL's position in the path. For example, in a scene with 6 VPLs and 3 light bounces there are 2 light paths in total and the 3rd VPL represents the second bounce in the first light path. To avoid replacing the first few VPLs every frame, a counter is kept such that the VPL replaced in the beginning of the frame is the invalid VPL after the last VPL replaced in the last frame. Before replacing a VPL, its parent VPL / light source is looked up to ensure that its valid. The new VPL is shot from the parent VPL's / light source's position to continue the light path.

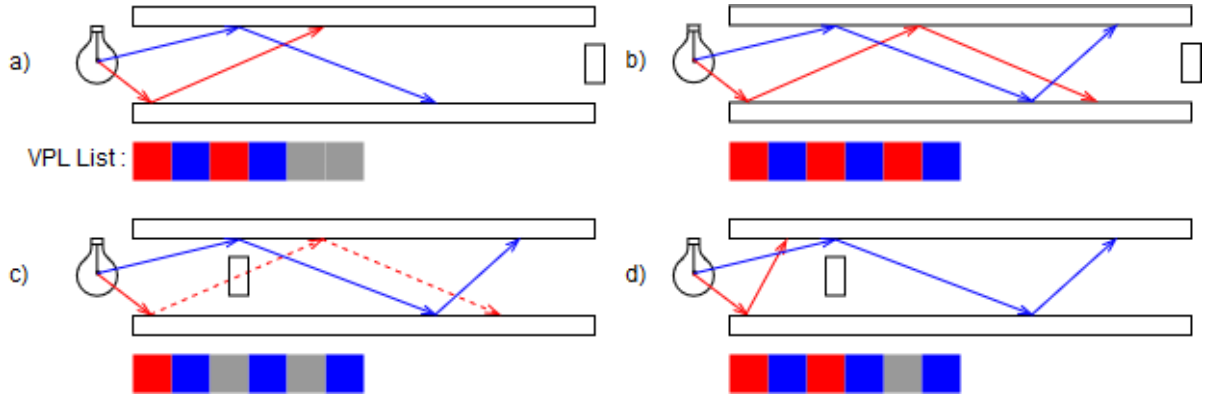



Figure 12: VPL Replacement

**Algorithm 3:** ReplaceVPLs() Pseudocode

```
lastVPLIndex = currentVPLIndex
VPLsToBeReplaced = {}
repeat
    currentVPLIndex = (currentVPLIndex + 1) % VPLs.size()
    VPL = VPLs[currentVPLIndex]
    parentVPL = GetParent(VPL, VPLs)
    if not IsValid(VPL) and IsValid(parentVPL) then
        VPLsToBeReplaced.insert(VPL)
    end
until currentVPLIndex != lastVPLIndex and VPLsToBeReplaced.size() <
 maxVPLsShotPerFrame
ReshootVPLs(VPLsToBeReplaced)
```

The scene is then rasterised from the camera's point of view (line 6). Position, normal, albedo and specularity buffers are filled to be used in the next stages. Since this operation is common in both direct and indirect shading, executing this method once saves time from redundant computation. The scene is also rasterised per light source to compute direct light shadow maps (line 7). Apart from shadow maps not being suitable for indirect shadows for performance reasons, direct shadows are computed separately from indirect shadows because they are the strongest shadows in a scene. This requires them to be computed every frame otherwise it would be very apparent that they are only updated periodically. After direct shadow maps, direct diffuse and specularity lighting is computed (line 8). Like the direct shadow map, this needs to be done every frame for all light sources, making use of the position, normal and specularity buffers.

To increase the number of simultaneous VPLs, the VPLs are split temporally across frames. This is done by cycling through sets of VPLs each frame. The sets are equally sized and predetermined in that if a VPL pool is split into 5 sets, every $5^{\text{th}}$ frame will have the same VPLs. This alone would result in a flickering image as different sets of VPLs are displayed each frame. This is dealt with further on in the pipeline.

**Algorithm 4:** VPL Temporal Sampling Pseudocode

```
currentTemporalIndex = (currentTemporalIndex + 1) % temporalSize
currentTemporalVPLSet = {}
for i = 0 to VPLs.size()/temporalSize do
    currentTemporalVPLSet.insert(VPLs[(temporalSize * i) +
     currentTemporalIndex])
end
```

For indirect shadows (line 9), occlusion tests are run between every pixel and VPL in the current temporal VPL set. The position buffer is used to determine each pixel's position. This prevents the need to run intersection tests from the camera's position to get the positions of the visible surfaces, improving performance. The number of occlusion tests needed is reduced via interleaved sampling. The results are saved as separate shadow buffers per VPL in the current temporal VPL set.

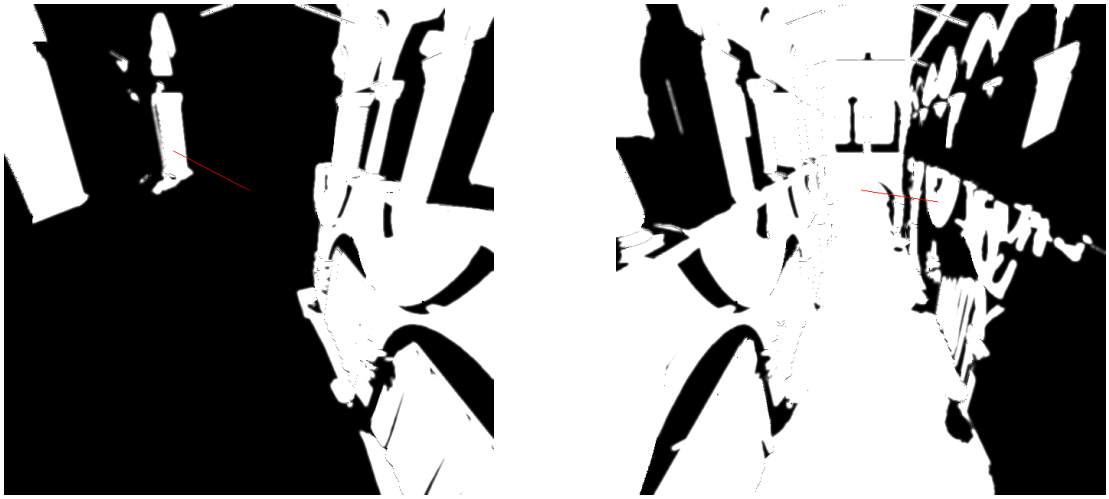


Figure 13: Indirect shadow buffers for different VPLs

Using the position and normal buffers, indirect diffuse lighting is calculated for the current temporal VPL set (line 10). The position buffer is also used to compute attenuation while the normal buffer is also used to implement the discontinuity buffer to fix the dithered output introduced by the indirect shadow stage. The indirect shadow buffers are applied as a mask on each VPL's lighting. The accumulation of each VPL's lighting is stored in the current temporal VPL set's frame buffer. The position buffer and scene transform are saved in a similar manner.

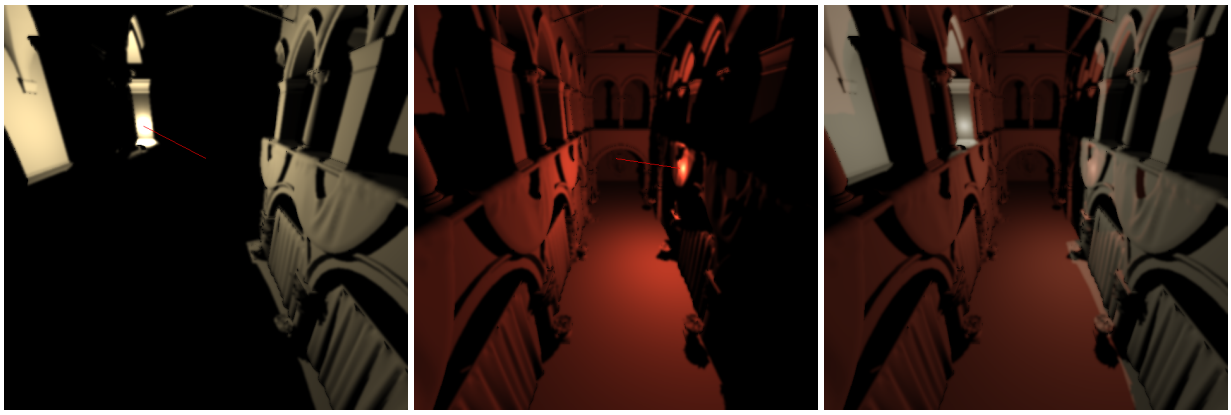


Figure 14: Accumulation of indirect lighting from different VPLs

To remove the flickering effect introduced by temporal sampling, a frame buffer per temporal VPL set is kept which stores the last output that was drawn using that particular VPL set. Each frame buffer is then re-projected to the current camera position so that the output is an accumulation of all frame buffers. Measures need to be taken when the previous frame buffers do not contain all surfaces present in the current frame buffer.

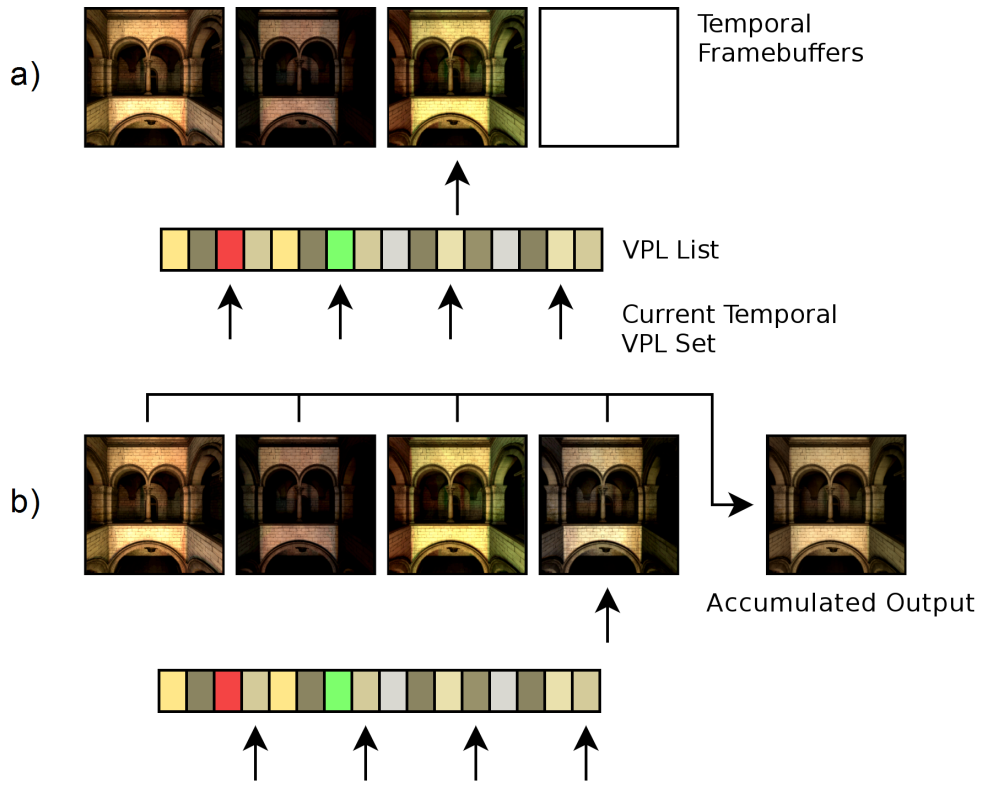



Figure 15: Temporal Sampling from one frame to the next

This is handled by the indirect re-projection stage (line 11). Re-projection is enabled using the position buffers and scene transform saved earlier. The scene transform is used to project world space co-ordinates from the current screen space to the previous frame buffer's screenspace. This allows the sampling of the correct pixel from the previous frame buffer despite a different scene transform. During re-projection on a per pixel basis, samples are discarded if their position does not match the surface they are being re-projected on. The discarding of samples is controlled with a cut-off to avoid aliasing from floating point errors. Setting the cut-off too high results in afterimages on previously occluded surfaces.

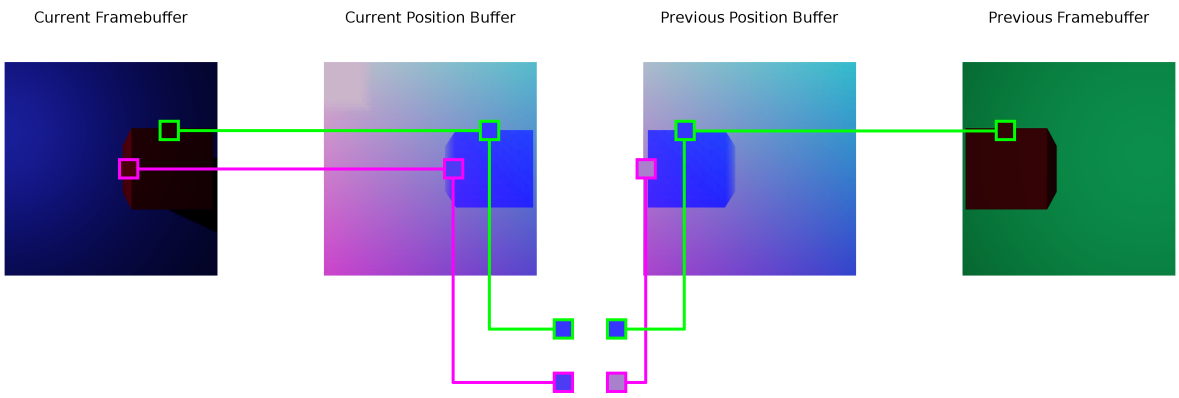



Figure 16: Temporal Sampling Example. In this scene the camera moves from the right to the left of a cube, this obscures the right side of the cube and reveals left side of cube from one frame to the other. Two VPLs are present on different coloured walls. The green sample, from the front of the cube, is valid while the purple sample, on the left of the cube, is invalid as the left side of the cube is not visible in the previous frame buffer.

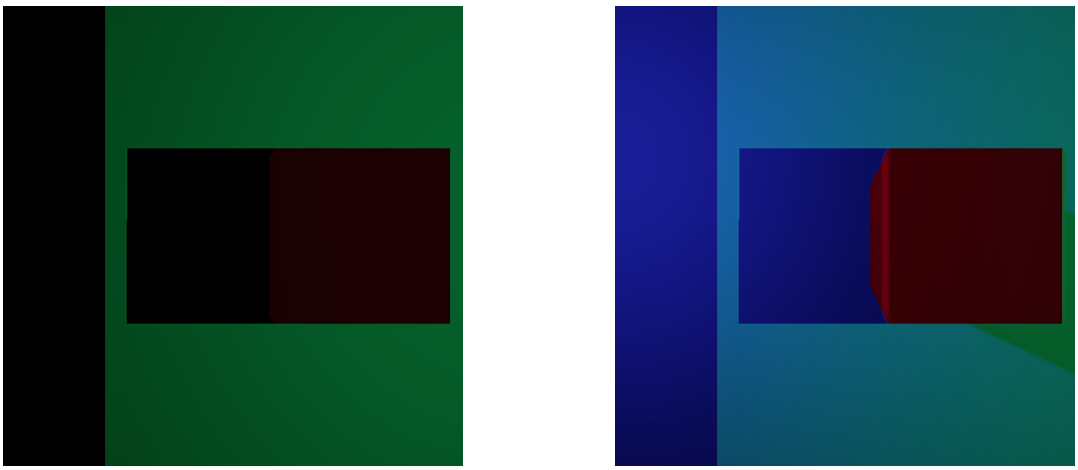


Figure 17: Re-projected frame buffer without invalid samples and the accumulated output.

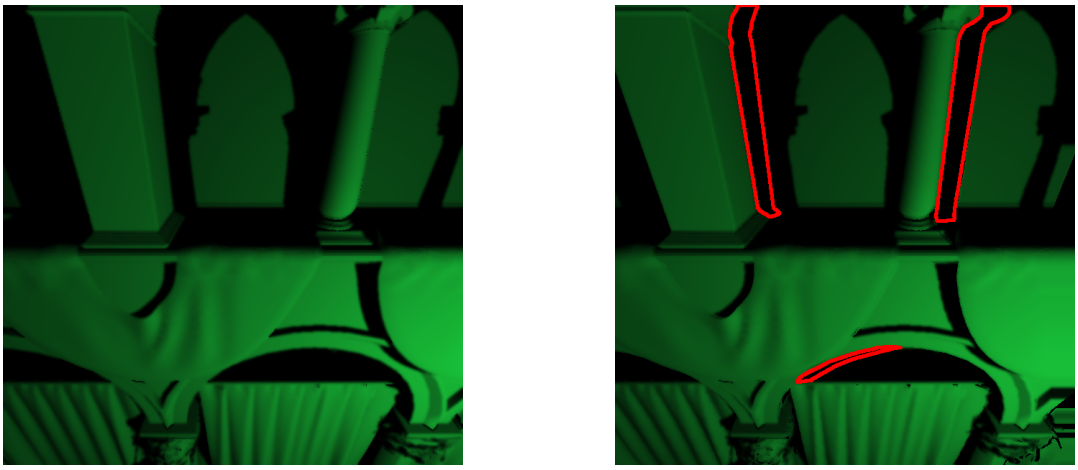


Figure 18: A temporal frame buffer and its re-projection, note how the previously occluded surfaces (outlined in red) are dark due to a lack of data.

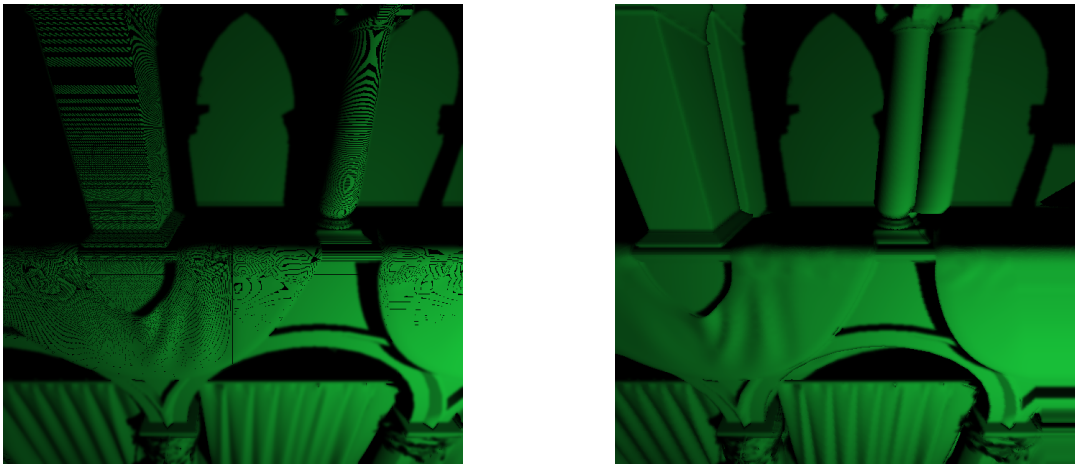


Figure 19: Low position cut-off aliasing vs high position cut-off afterimages

Finally, the results from the direct shading and indirect re-projection stages are combined and drawn to the screen (line 12). The albedo buffer is applied here to preserve detail after the re-projection.

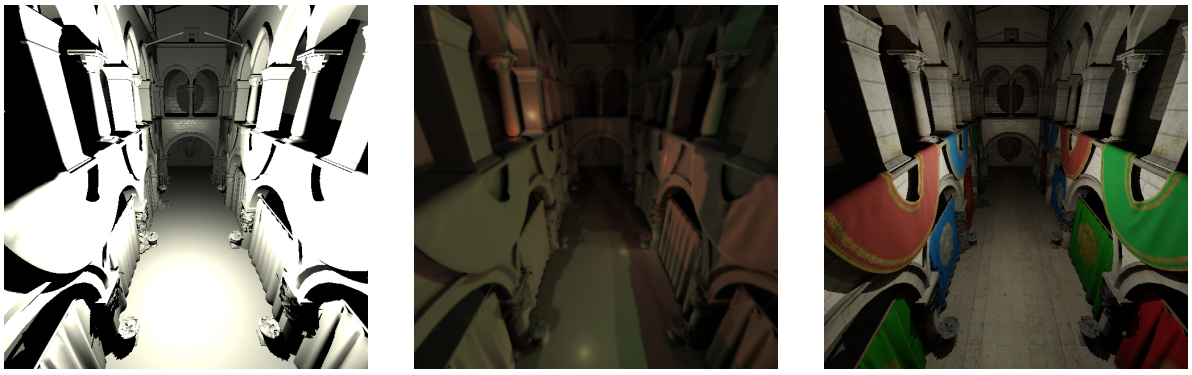


Figure 20: Direct Lighting, Indirect Lighting (12 VPLs) and their combination with the albedo buffer

This design was developed concurrently with the implementation, and thus was influenced by underlying structures in the implementation. An example of this would be the separation of the VPL Invalidation and VPL Replacement stages. Running invalidation checks one by one as VPLs are iterated over to be replaced would perform poorly compared to running all invalidation checks at once due to overhead costs of executing ray-primitive tests. This is expanded upon in the next section.

# 5 Implementation

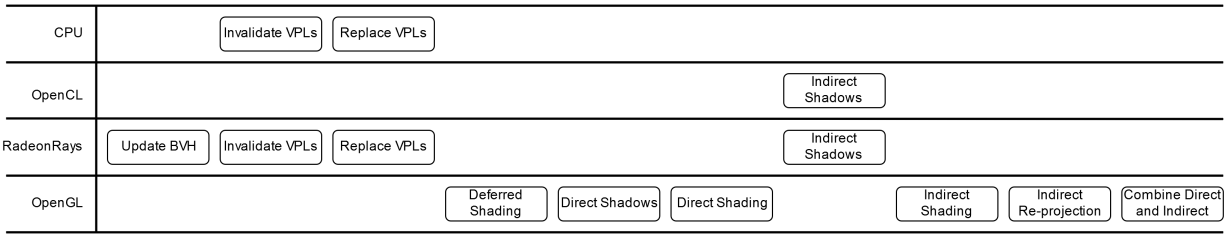



Figure 21: Frame Stages Implementation

The implementation revolves around 3 libraries, OpenGL, OpenCL and Radeon Rays [21]. OpenGL is an open, portable library that enables hardware accelerated 3D graphics using the GPU. OpenCL is the general computing counterpart of OpenGL, allowing execution of programs on the GPU that aren't related to 3D graphics. Admittedly Vulkan is a newer library that aims to replace both OpenGL and OpenCL, however it does so at a lower level of abstraction. While this enables the possiblity for faster performance, Vulkan introduces more complexity and less portability. This would increase development time

required for the implementation, which was limited. Radeon Rays runs on top of a general compute library like OpenCL and Vulkan to provide hardware accelerated ray-primitive tests. Internally Radeon Rays uses the two-level BVH described earlier (Background - Acceleration Structures). It provides two types of tests, intersection and occlusion tests. Given an origin and direction, intersection tests return the closest hit surface. Specifically, intersection tests return the index of the face hit by the ray and barycentric co-ordinates to identify exactly where on the face the ray has hit. Occlusion tests return as soon as a ray hits any surface between its origin and destination. This saves time compared to intersection tests.

Initialisation involves linking OpenGL and OpenCL sessions so that buffers can be shared between the two libraries, loading the scene from files and allocating memory as necessary. Since initialisation is only done once, performance in this stage is not a concern. Although invalidating and replacing VPLs would benefit from being run entirely on the GPU, these stages are given less priority since indirect shadows are expected to be much more taxing, thus a hybrid approach is used for simplicity. Occlusion tests for invalidating VPLs are configured on the CPU before being executed on the GPU. The results are then read and VPLs are invalidated accordingly on the CPU. The same happens when replacing VPLs where intersection tests for shooting new VPLs are configured on the CPU prior to being run all at once on the GPU. The face index is used to lookup the material associated with the face and the barycentric co-ordinates are used to sample the correct pixel from any textures applied to the face when merging properties for new VPLs on the CPU.

Indirect shadows use occlusion tests. The tests are configured using OpenCL on the GPU due to the large amount of tests needed per VPL. This also avoids memory bottlenecks having to transfer data from the CPU to the GPU. Mapping the results to VPL shadow buffers is also done this way. While saving shadow buffers separately per VPL is memory inefficient, this allows GPU threads to be completely independent from each other. This is due to each thread reading a unique occlusion test result and writing to a unique location, preventing the need for thread synchronisation, improving performance.

The rest of the stages are done using shaders in OpenGL. Each stage is represented by a different shader. While this choice introduces potential overhead costs from having to load different shaders to be executed in the GPU, it prevents code becoming too convoluted, aiding development. The buffers used are stored on the GPU, also preventing the memory bottleneck described earlier. As more VPLs are present, memory is used up. If 1024x1024 resolution is used, each shadow map is at least $1024^2$ bits large, this can become a problem

with more than 500 VPLs. Using interleaved sampling it is possible to reduce indirect lighting buffers' size since a lower resolution is used per VPL. Temporal sampling also helps by allowing the reuse of VPL shadow buffers for different VPL sets. For example, using 25 VPLs with 5 temporal sampling frame buffers, only 5 shadow buffers are going to be used each frame. This is done by mapping VPLs to shadow buffers by their arbitrary position in the current temporal VPL set.

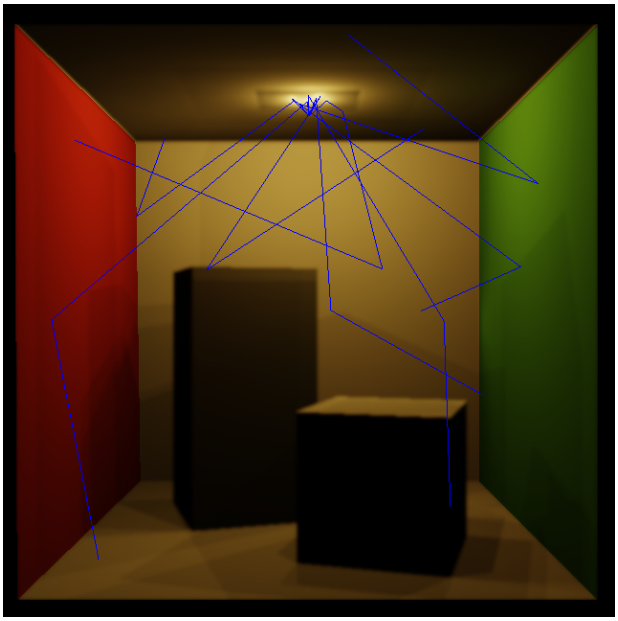


Figure 22: VPL Debug Visualisation

Debugging is a large part of any development project. To facilitate debugging, a shader was written to display the paths taken by each VPL inside the interactive session, enabled by the press of a button. This allows visualisation of any VPL invalidation or shooting issues. Each VPL's light contribution can then be cycled through individually. This was particularly useful in understanding issues experienced when sharing buffers between OpenGL and OpenCL, shedding light on what was actually happening underneath the high level method calls between the two libraries.

# 6 Results

In this section, the implementation developed is evaluated in two aspects, performance and quality. The choices made behind each test is detailed out, the requirements for success are outlined, predictions are made, and test results compared.

## 6.1 Performance

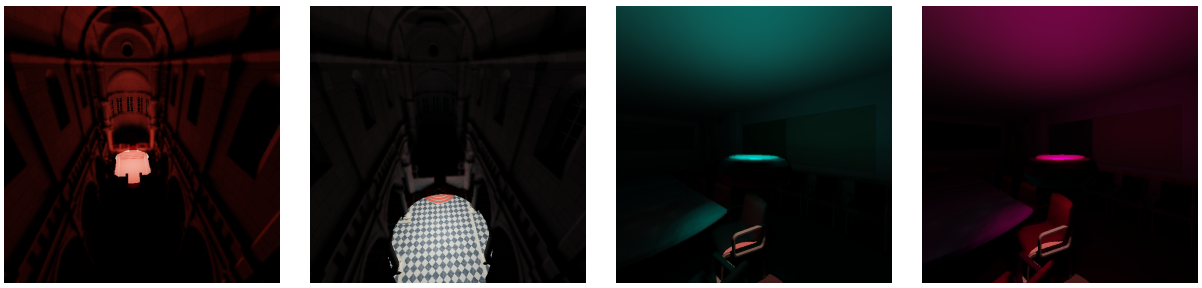


Figure 23: Sibenik, Conference Scenes in the beginning and end of their tests

For temporal sampling to be viable, it has to return faster frame times than without using temporal sampling. The implementation developed throughout this dissertation supports interactive sessions, such as allowing the user to move the camera and light source in real-time. However to measure the difference between running with and without temporal sampling, the implementation also supports running pre-determined animations to keep the situations encountered constant, for example VPL reshooting and BVH updates. Beside frame times, time intervals between the stages in the pipeline were also recorded. The time intervals allow the identification of the stages that bottleneck the pipeline throughout various scenarios. The following scenes [22] were configured for testing performance:

- Cornell Box, Sponza and Crytek Sponza are static scenes with increasing geometric complexity. In these scenes, the light source does not move, and the camera does not move either because the method shoots the same number of shadow rays independent of camera movement. For this reason, Indirect Shadows is expected to be the most taxing stage as no VPLs are invalidated after the scene is initialised - leaving VPL Replacement inactive for most of the running time.

- In the Sibenik scene a spotlight moves across the floor from the carpet on to the marble. The sibenik is mostly illuminated by indirect lighting, due to the direct lighting being focused downwards. This scene will test which stage is the most taxing between Indirect Shadows and VPL Replacement, since VPLs are constantly being invalidated at the edges of the spotlight as it moves forward, forcing the VPL Replacement stage to be more active than in the static scenes.

- The Conference scene tests the dynamic aspect of the method by having a spotlight on top of a rotating donut. The donut is coloured turquoise on one side and purple

on the other, as the donut's top side changes, so does the overall lighting in the scene. While rotating, VPLs on the donut are occluded by the donut itself, regularly invalidating the VPLs, this taxes the VPL Replacement stage. More importantly when scene geometry changes, the BVH tree must be updated appropriately, this scene tests whether the BVH can be updated in real-time.

Each scene was tested at 1024x1024 resolution with 512 VPLs using 4x4 interleaved sampling and varying levels of temporal sampling. The tests were run on a system with an AMD FX-8350 CPU and RX Vega 56 GPU. As evidenced by figure 24's positive results, frame times are lowered with the increase of temporal frame buffers.

| Scene | Temporal Sampling Size | | | | |
|---|---|---|---|---|---|
| | 1 | 4 | 8 | 16 | 32 |
| cornell box | 271.948 | 72.636 | 41.226 | 27.416 | 25.159 |
| sponza | 554.105 | 135.938 | 70.217 | 42.177 | 37.376 |
| crytek sponza | 489.522 | 126.023 | 71.443 | 51.522 | 47.228 |
| sibenik | 464.277 | 120.021 | 65.575 | 41.214 | 37.010 |
| conference | 453.456 | 118.945 | 71.721 | 60.561 | 46.951 |

Figure 24: Average Frame Times at 512 VPLs (ms)

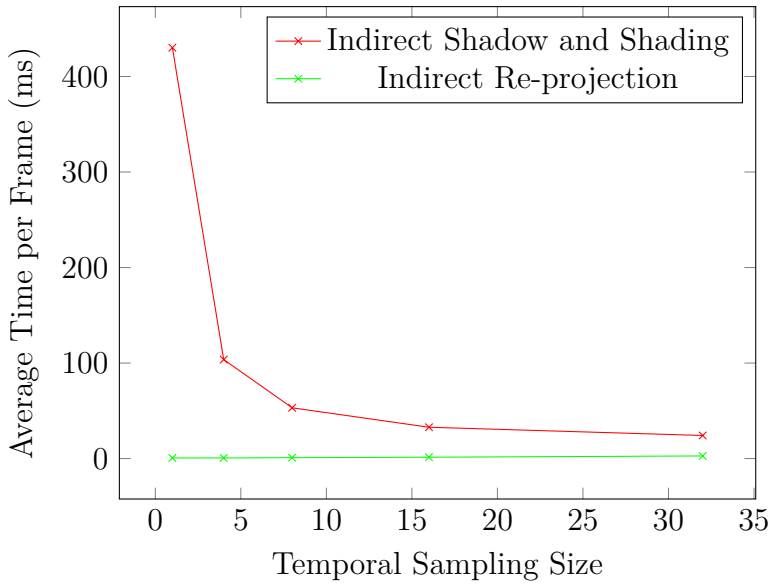



Figure 25: Average Time per Indirect Stage vs Temporal Sampling Size

Figure 25 shows how temporal sampling performance costs do not grow as fast as VPL shading costs. This is what enables the performance benefits shown in Figure 24. Checking the stage interval times (figure 26) shows that the computation of indirect shadows is the most taxing stage across all scenes. The Indirect Shadows stage is dependent on geometry complexity as evidenced by the faster time in Cornell Box. VPL Replacement is not as taxing as Indirect Shadows despite using intersection tests instead of occlusion tests. This is due to the sheer amount of occlusion tests needed since Indirect Shadows is dependent on screen resolution. As expected the BVH Construction stage is only active in the Conference scene since it is the only scene in which geometry moves. The low amount of time taken to update the BVH shows that it can be updated in real-time.

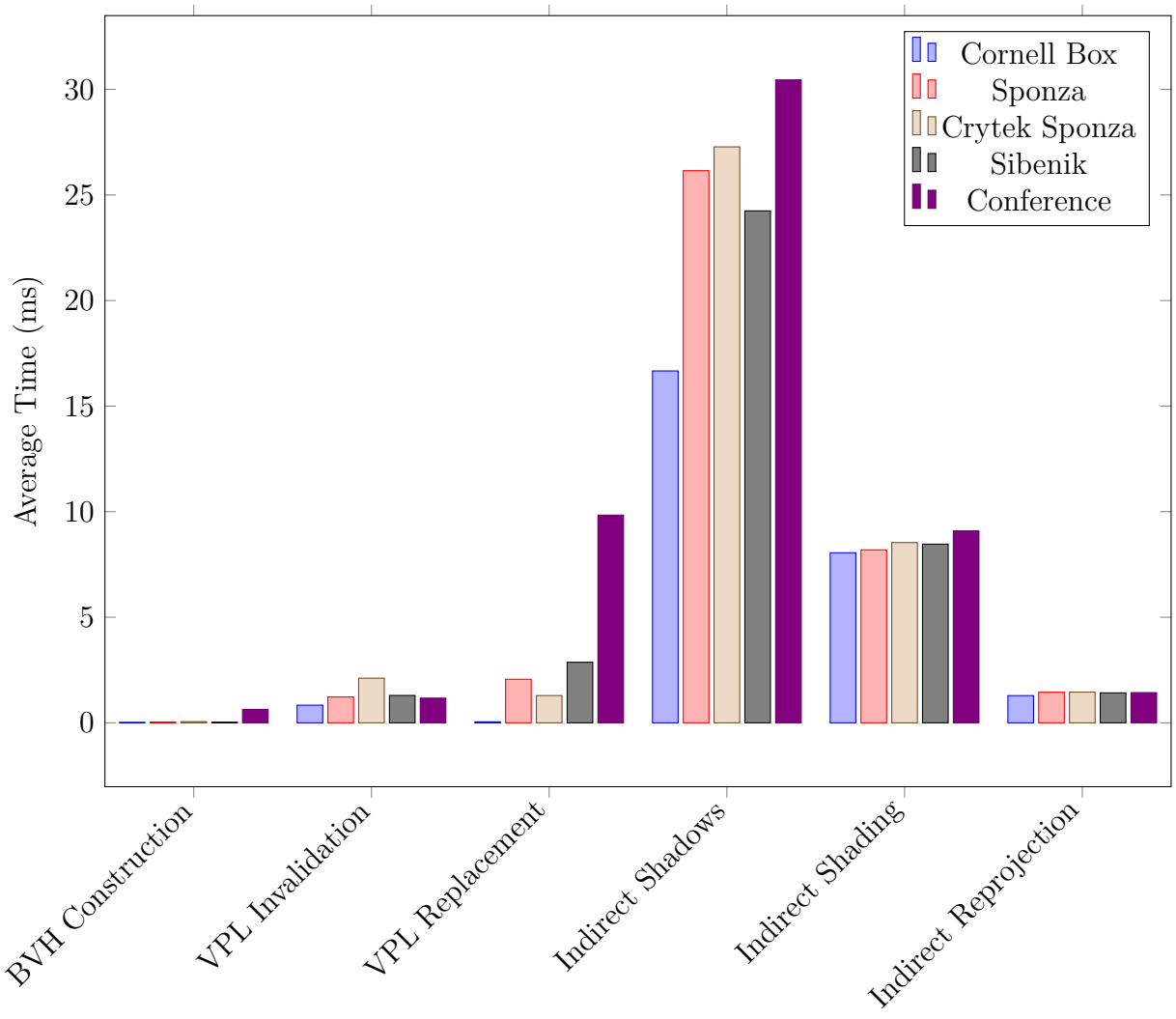



Figure 26: Average Time per Stage at 16 Temporal Frame Buffers

## 6.2 Quality

Since temporal sampling lowers the number of VPLs being rendered each frame, temporal sampling increases time taken for lighting to converge on previously occluded surfaces. During convergence the surfaces missing from temporal frame buffers are not properly lit by the VPLs associated to the frame buffers. For example taking the Sponza scene at around 27 frames per second (fps) with 32 temporal frame buffers, it would take more than a second for the scene to be completely re-rendered, which would be noticeable. On the other hand, the same scene with 16 temporal frame buffers operates at 24 fps and takes around 0.6 seconds for lighting to completely converge. In this example the user may find using 16 temporal frame buffers is a smoother experience despite the slightly lower fps. Therefore one needs to takes this into account when choosing the temporal sampling size, depending on the dynamic nature of the scene. The screenshots in figure 27 showcase the artefacts that are visible for a limited amount of time due to this issue. Turning darkens the image from the direction the user is turning, this can be seen in the turning downwards example. Foreground objects in the foreground appear to cast shadows on background objects due to surfaces that become visible due to movement and are not present in any of the temporal frame buffers, this can be seen in the strafing example. Note that these examples were captured with 32 temporal frame buffers to exaggerate the artefacts to be able to showcase them.

Figure 27: Temporal Sampling Artefacts : Static, Turning Downwards, Strafing Right

Beside the artefacts present in dynamic scenes, temporal sampling should not affect image quality given lighting is allowed to converge on any previously occluded surfaces. To ensure that this is true, screenshots with and without temporal sampling were saved from the implementation to be compared with images from a reference renderer, in this

case - Mitsuba [23]. The images were compared via HDR-VDP-2 [24], a visual metric that measures human perceivable differences between images. Cornell Box and Sponza were chosen for this test as they demonstrate some phenomena introduced by global lighting that are important to be preserved when applying temporal sampling. The Cornell Box exhibits color mixing, the boxes in the middle aren't coloured however by indirect light reflecting of the coloured walls, the boxes' sides should be coloured red and green respectively. Sponza features many occluded areas from direct light, indirect light should prevent these areas from being pitch dark.

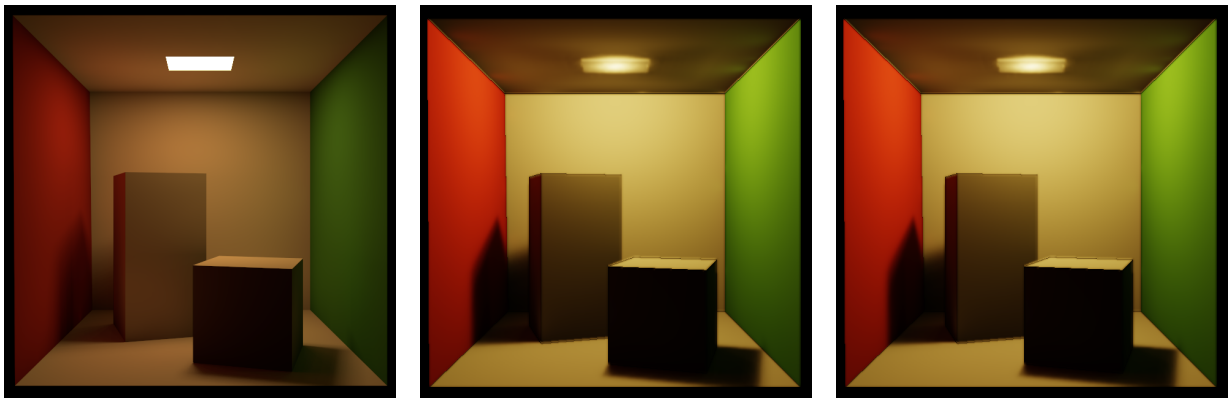


Figure 28: Cornell Box : Reference Image, With and Without Temporal Sampling

Figure 29: Sponza : Reference Image, With and Without Temporal Sampling

In the quality comparison tests, the reference image is very different from the implementation's output. This is because the VPLs inside the implementation are not shot exactly in the same directions as the reference renderer, leading to different lighting. However the difference in quality between the reference renderer and implementation is constant with

| Scene | With Temporal Sampling | Without Temporal Sampling |
|---|---|---|
| cornell box | 55.4427 | 55.4808 |
| sponza | 49.1801 | 49.1423 |

Figure 30: HDR-VDP-2 Scores

and without temporal sampling as shown by the very similar HDR-VDP-2 scores. Therefore temporal sampling does not introduce any other artefacts besides the ones already described in figure 27.

The average fps between all scenes reached was 26 fps, an impressive feat considering that base performance without temporal sampling was 0.5 fps. The fps can always be increased by decreasing the total number VPLs used, altough the benefits of a large VPL count will obviously be reduced. Quality has also been confirmed to be maintained apart from minor artefacts.

# 7 Conclusions and Future Work

This dissertation aimed to achieve faster performance on instant radiosity algorithms for diffuse surfaces using hardware accelerated ray-primitive tests with minimal quality detriments via a method that makes use of temporal filtering. This required the development of a system with the implementation of this method to allow proper evaluation of the method. As shown by the results, the method is successful in this regard. By reusing illumination information on previous frames, temporal sampling is able to achieve up to 15x speedup with temporary artefacts as any missing illumination information from previous frames is filled in. 40 fps in the Cornell Box scene is the best frame rate achieved at 512 VPLs with 1024x1024 resolution by this method in the performance tests. The re-projection used in this method has been shown to be cheaper than indirect lighting computation. Therefore this method is likely to scale well with better hardware since the largest bottleneck in the pipeline is the large amount of occlusion tests needed for indirect lighting shadows.

## 7.1 Future Work

The method currently renders a VPL set each frame from scratch regardless of how much of the frame buffer associated with the VPL set is valid to be reused. Future work can expand on this by selectively rendering parts of a scene depending on the amount of valid

data available. Although as with re-projection, one would have to take care that searching for valid data is not costlier than rendering the entire scene. This could be extended further by selectively updating all frame buffers every frame to eliminate any artefacts introduced by the lack of valid data.

# References


[1] G. Moran and M. Leo, "The 'reflections' ray-tracing demo presented in real time and captured live using virtual production techniques," in *ACM SIGGRAPH 2018 Real-Time Live!*, ser. SIGGRAPH '18. New York, NY, USA: ACM, 2018, pp. 8:1–8:1. [Online]. Available: http://doi.acm.org/10.1145/3229227.3229230

[2] L. Williams, "Casting curved shadows on curved surfaces," *SIGGRAPH Comput. Graph.*, vol. 12, no. 3, pp. 270–274, Aug. 1978. [Online]. Available: http://doi.acm.org/10.1145/965139.807402

[3] T. L. Kay and J. T. Kajiya, "Ray tracing complex scenes," in *SIGGRAPH*, 1986.

[4] B. Segovia and M. Ernst, "Memory efficient ray tracing with hierarchical mesh quantization," in *Proceedings of Graphics Interface 2010*, ser. GI '10. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2010, pp. 153–160. [Online]. Available: http://dl.acm.org/citation.cfm?id=1839214.1839242

[5] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, no. 6, pp. 343–349, Jun. 1980. [Online]. Available: http://doi.acm.org/10.1145/358876.358882

[6] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 137–145, Jan. 1984. [Online]. Available: http://doi.acm.org/10.1145/964965.808590

[7] J. T. Kajiya, "The rendering equation," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 143–150, Aug. 1986. [Online]. Available: http://doi.acm.org/10.1145/15886.15902

[8] D. P. Greenberg, M. F. Cohen, and K. E. Torrance, "Radiosity: A method for computing global illumination," *The Visual Computer*, vol. 2, pp. 291–297, 1986.

[9] D. S. Immel, M. F. Cohen, and D. P. Greenberg, "A radiosity method for non-diffuse environments," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 133–142, Aug. 1986. [Online]. Available: http://doi.acm.org/10.1145/15886.15901

[10] A. Keller, "Instant radiosity," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 49–56. [Online]. Available: https://doi.org/10.1145/258734.258769

[11] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek, "Interactive global illumination using fast ray tracing," in *Proceedings of the 13th Eurographics Workshop on Rendering*, ser. EGRW '02. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 15–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=581896.581899

[12] C. Benthin, I. Wald, and P. Slusallek, "A scalable approach to interactive global illumination," *Comput. Graph. Forum*, vol. 22, pp. 621–631, 2003.

[13] B. Segovia, J. C. Iehl, R. Mitanchey, and B. Péroche, "Bidirectional instant radiosity," in *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, ser. EGSR '06. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 389–397. [Online]. Available: http://dx.doi.org/10.2312/EGWR/EGSR06/389-397

[14] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila, "Incremental instant radiosity for real-time indirect illumination," in *Rendering Techniques*, 2007.

[15] I. Radax, "Instant radiosity for real-time global illumination," 2008.

[16] H. W. Jensen, "Global illumination using photon maps," in *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. London, UK, UK: Springer-Verlag, 1996, pp. 21–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=275458.275461

[17] M. McGuire and D. Luebke, "Hardware-accelerated global illumination by image space photon mapping," in *Proceedings of the Conference on High Performance Graphics 2009*, ser. HPG '09. New York, NY, USA: ACM, 2009, pp. 77–89. [Online]. Available: http://doi.acm.org/10.1145/1572769.1572783

[18] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ser. I3D '10. New York, NY, USA: ACM, 2010, pp. 99–107. [Online]. Available: http://doi.acm.org/10.1145/1730804.1730821

[19] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing: A preview," in *Symposium on Interactive 3D Graphics and Games*, ser. I3D '11. New York, NY, USA: ACM, 2011, pp. 207–207. [Online]. Available: http://doi.acm.org/10.1145/1944745.1944787

[20] B. Walter, G. Drettakis, and S. G. Parker, "Interactive rendering using the render cache," in *Rendering Techniques*, 1999.

[21] "Introducing the radeon rays sdk," Tech. Rep., 2016. [Online]. Available: https://32ipi028l5q82yhj72224m8j-wpengine.netdna-ssl.com/wp-content/uploads/2016/08/169798-A$_A MD_R adeon Rays_I ntro_F NL.pdf$

[22] M. McGuire. (2017, July) Computer graphics archive. [Online]. Available: https://casual-effects.com/data

[23] W. Jakob, "Mitsuba renderer," 2010, http://www.mitsuba-renderer.org.

[24] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich, "Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions," *ACM Trans. Graph.*, vol. 30, pp. 40:1–40:14, 2011.